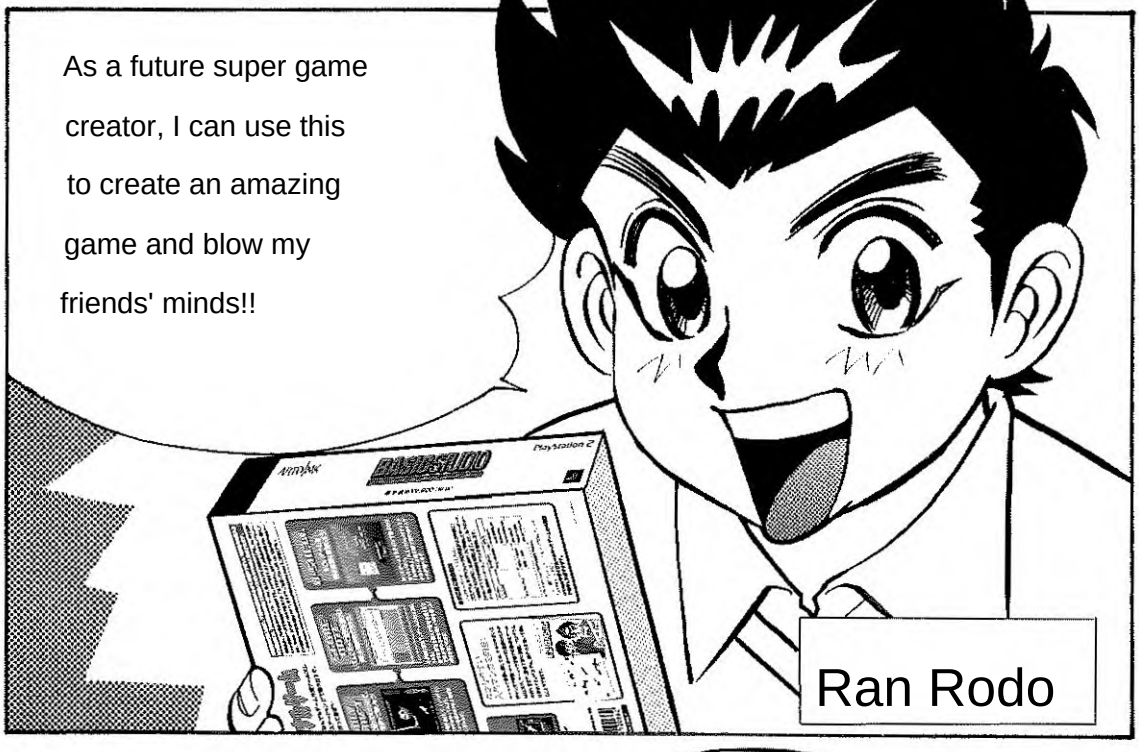
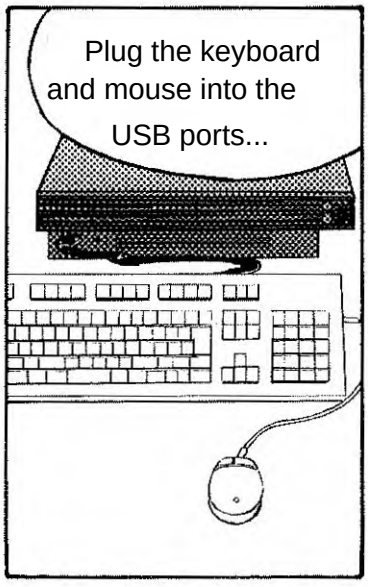
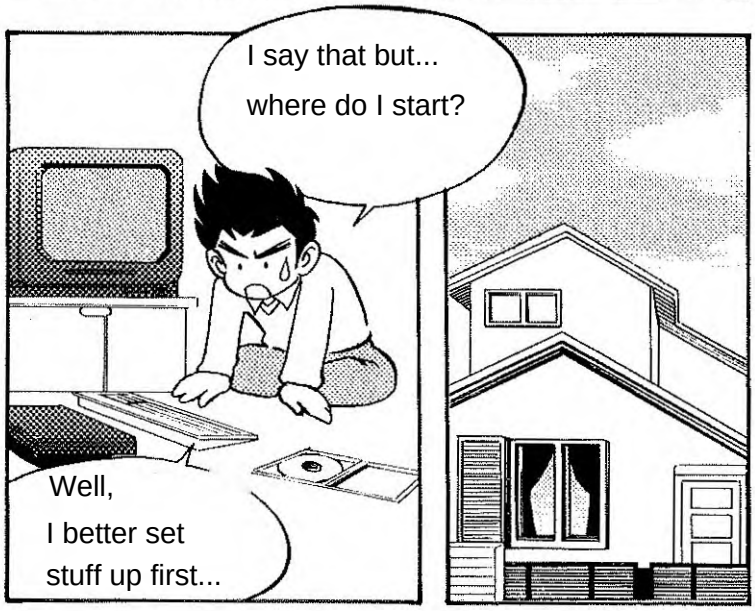
はじめよう！

EZ-BASIC

Ch. 1



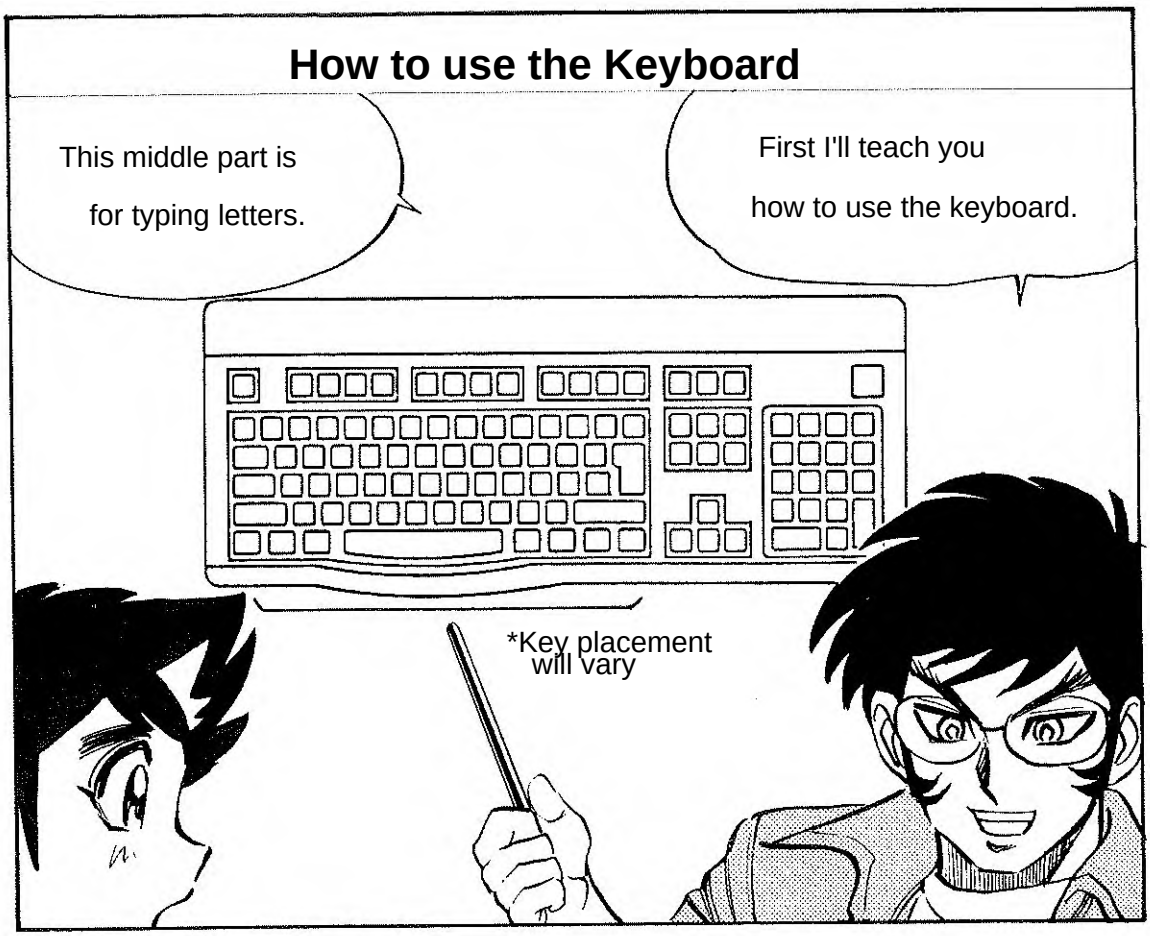


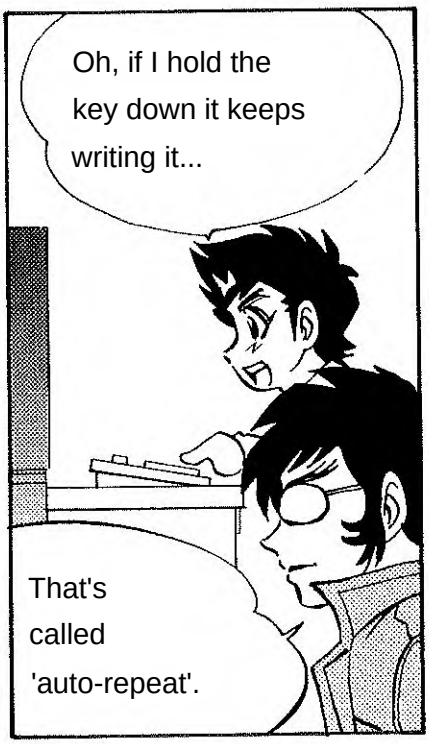


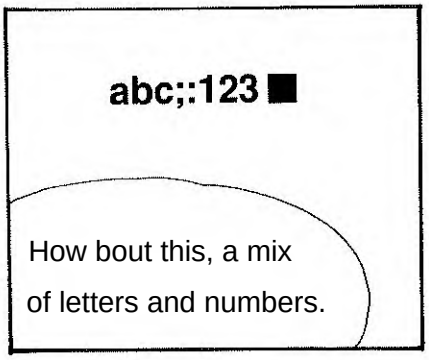
*The same operations can be performed with an analog controller (DUALSHOCK 2).

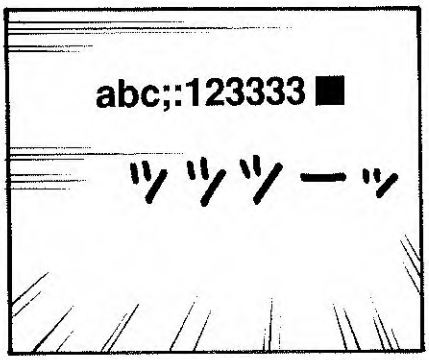
EZ-Basic is for making a game but this other stuff is for drawing and making music I think...
B-SHELL
Browsing
POLY-CRAFT
3D MODELING
OPTION
CONFIGURATION
I'm making a game so I'll boot EZ-BASIC.
In goes disc 1...
ウィィィン
チカ
チカ
What is this? It's just a blinking square...
You're just in time, teach me BASIC!
Hmph... That's why I followed you from the store.
You're struggling huh, Ran.
ガーン
Oh! Mr. Tsuyoshi!!

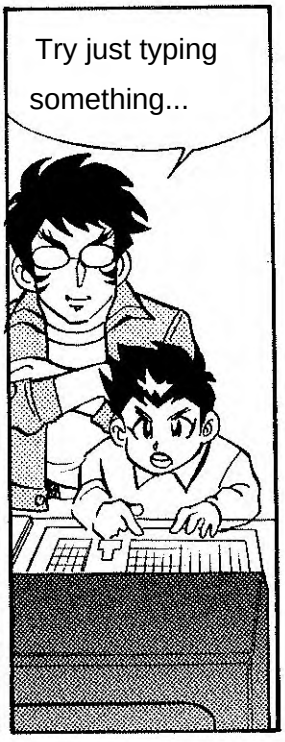
How to use the Keyboard





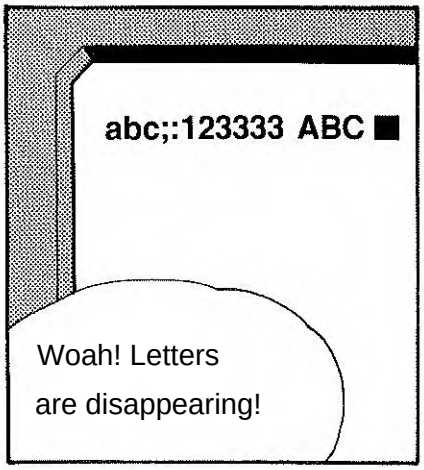


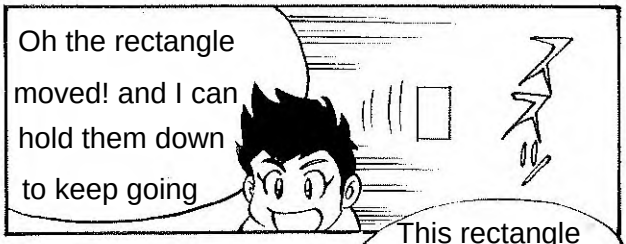


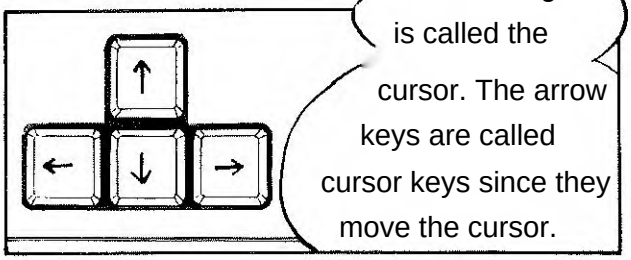


Some keys have several symbols on them at the top of the keyboard.
In the middle there's this big blank key...
That's the spacebar. Use it to add spaces.
Hold down shift and press one of those...
Try holding shift and pressing those keys.
ぽち ぽち
Oh it made capital letters! And I can write those symbols that are above the numbers!
abc;:123333 ABC!"#$% ■
Next try pressing Back Space in the top right.
Enter
Oh the cap lock light lit up.
This way you don't have to hold shift to write capitals.
Press caps lock while holding shift.

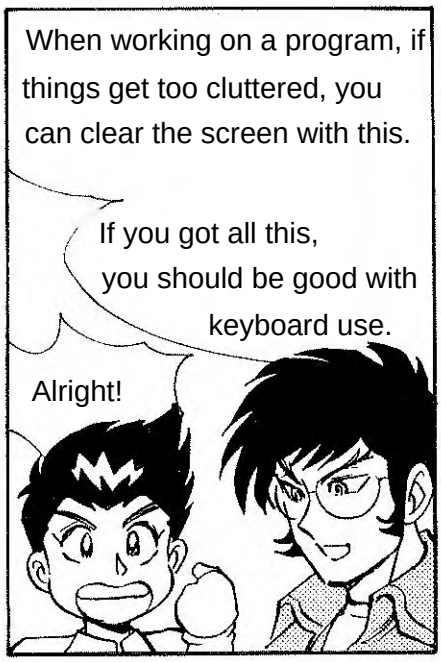
That key lets you go back one space and delete.
If you hold it down, auto-repeat will clear out all that you have written.

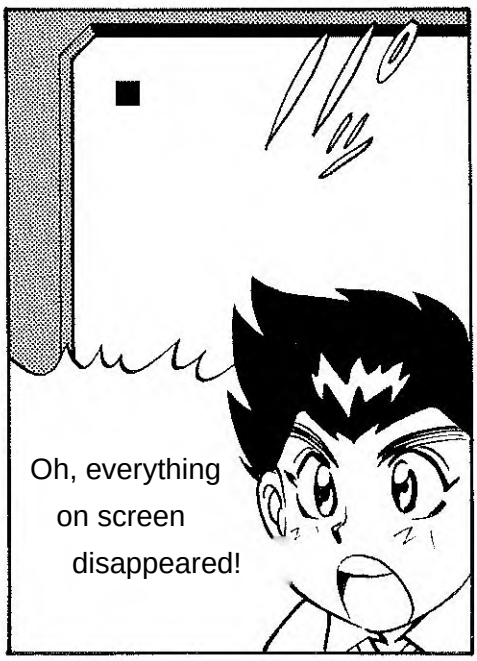

abc;:123333 ABC
Woah! Letters are disappearing!

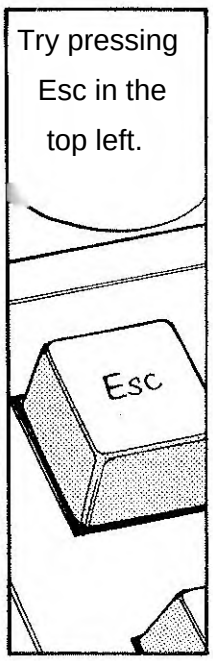

Oh the rectangle moved! and I can hold them down to keep going

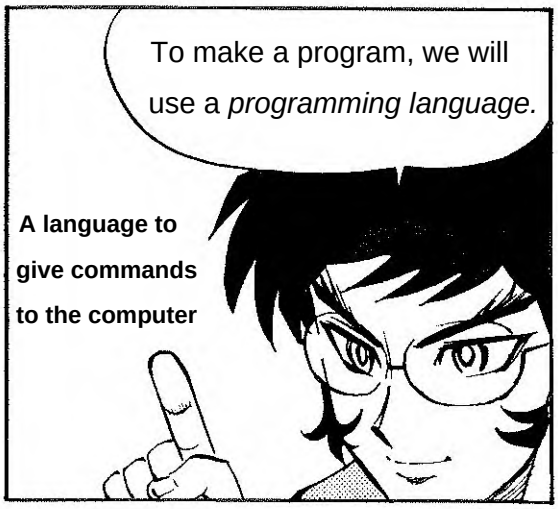

This rectangle is called the cursor. The arrow keys are called cursor keys since they move the cursor.

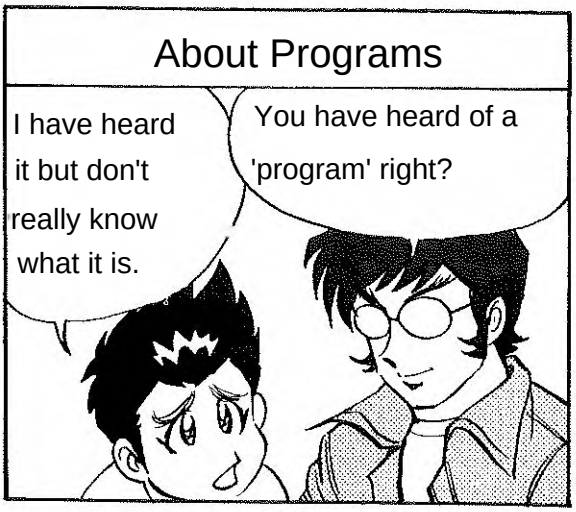
Next try the keys with arrows on them.
Alright!

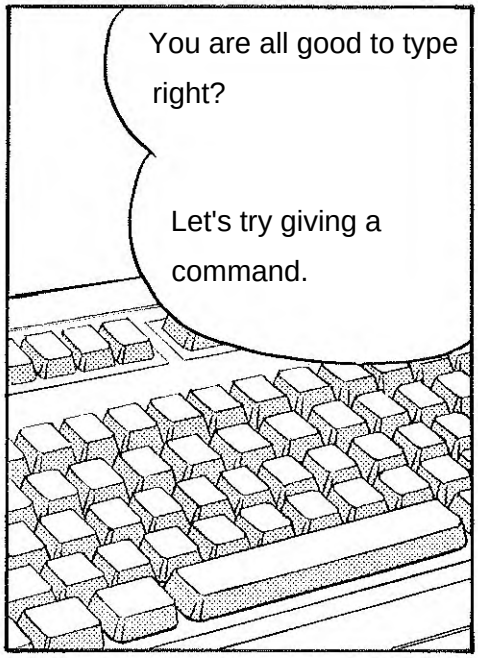

When working on a program, if things get too cluttered, you can clear the screen with this.
If you got all this, you should be good with keyboard use.
Alright!


Oh, everything on screen disappeared!


Try pressing Esc in the top left.
Esc







Machine language

A language that expresses meaning through combinations of 0 and 1. Programs are rarely written directly in this language.

Assembly

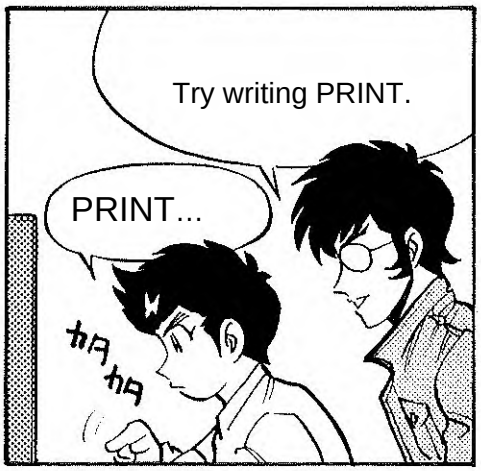
A language that replaces machine code with simple words. It only has simple instructions, but can execute quickly.

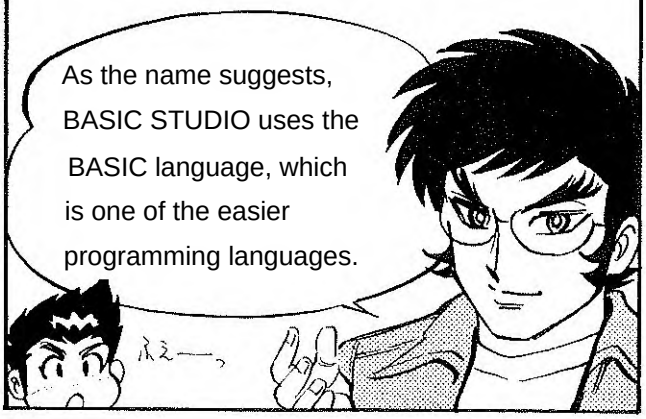
C/C++

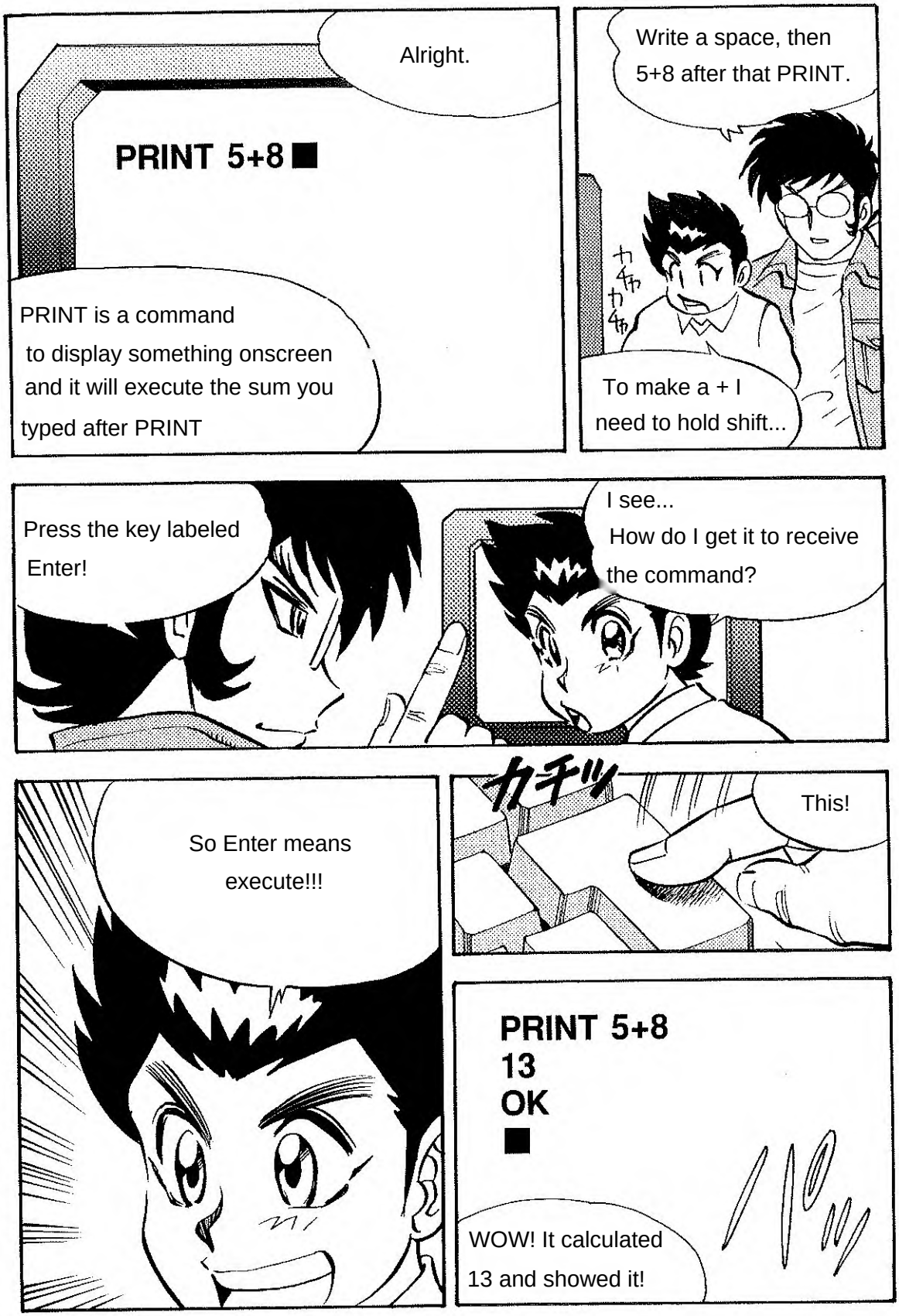
The most commonly used language today. Although some parts are difficult for beginners to understand, it is highly versatile.

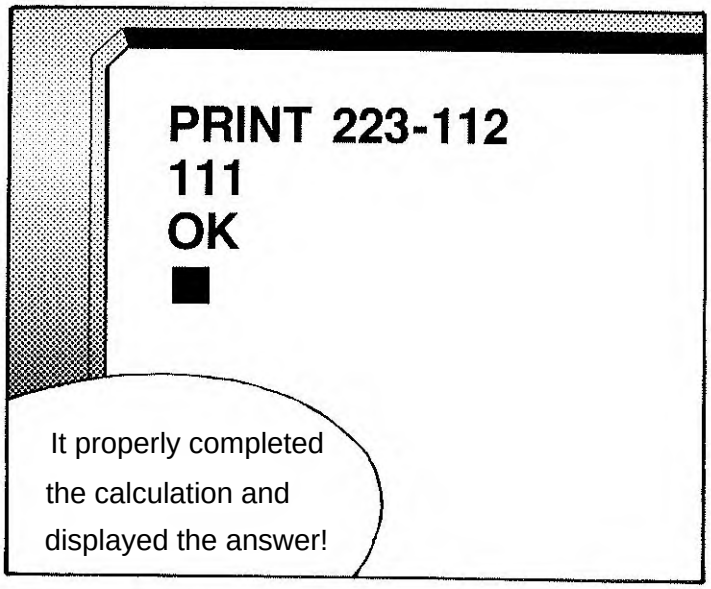
BASIC

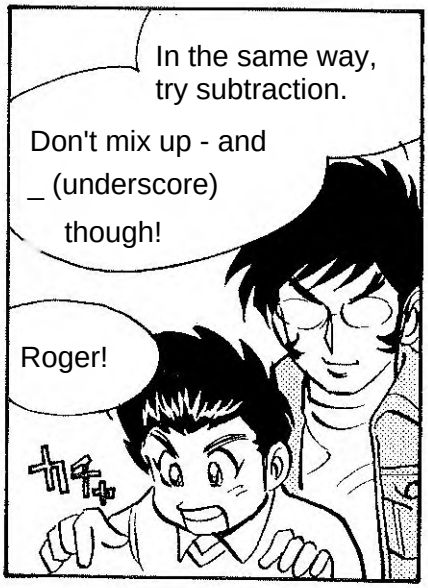
A language designed to be easy to understand even for beginners. It is easy to find and correct errors in programs.



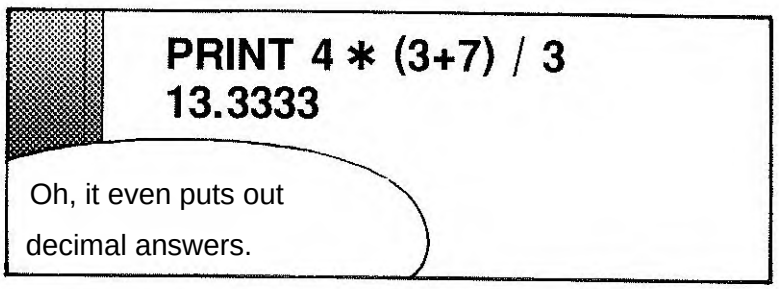


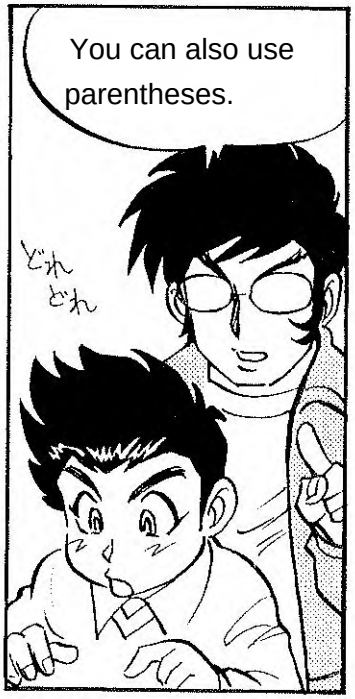

Alright.
PRINT 5+8■
PRINT is a command to display something onscreen and it will execute the sum you typed after PRINT
Write a space, then 5+8 after that PRINT.
カチカチ
To make a + I need to hold shift...
Press the key labeled Enter!
I see... How do I get it to receive the command?
So Enter means execute!!!
カチッ
This!
PRINT 5+8
13
OK
■
WOW! It calculated 13 and showed it!

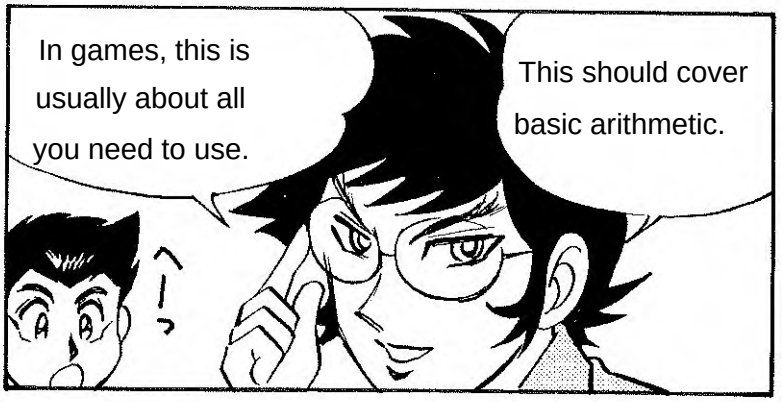

PRINT 223-112
111
OK
■
It properly completed the calculation and displayed the answer!


In the same way, try subtraction.
Don't mix up - and _ (underscore) though!
Roger!
カチャ

Multiplication uses a *
Division uses a /


PRINT 4 * (3+7) / 3
13.3333
Oh, it even puts out decimal answers.


You can also use parentheses.
どれ
どれ


In games, this is usually about all you need to use.
This should cover basic arithmetic.
!?

10 PRINT 20 * 30

Just like this...

Huh, now it doesn't show the answer...

Execute with Enter…

Inputting a Program

Next are programs.

Try putting a 10 in front of the command you just wrote.

Roger.

After writing, press the Enter key.

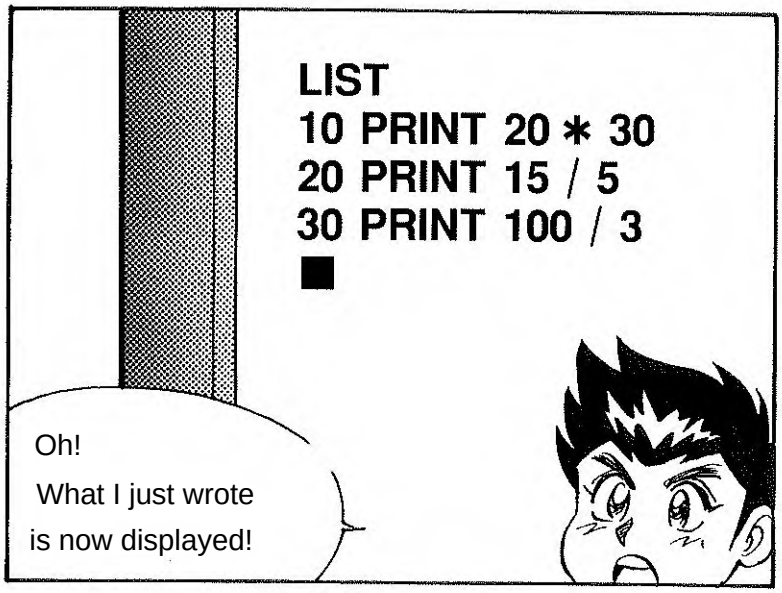
In the same way, write commands with the numbers 20 and 30.

And the instructions with line numbers are recorded as a program!!!

The number written before a command is called the line number.

10 PRINT 20 * 30
20 PRINT 15 / 5
30 PRINT 100 / 3
■

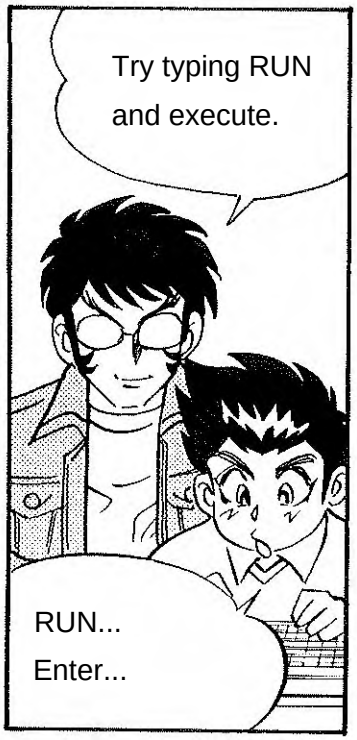
Yeah, still nothing happens.

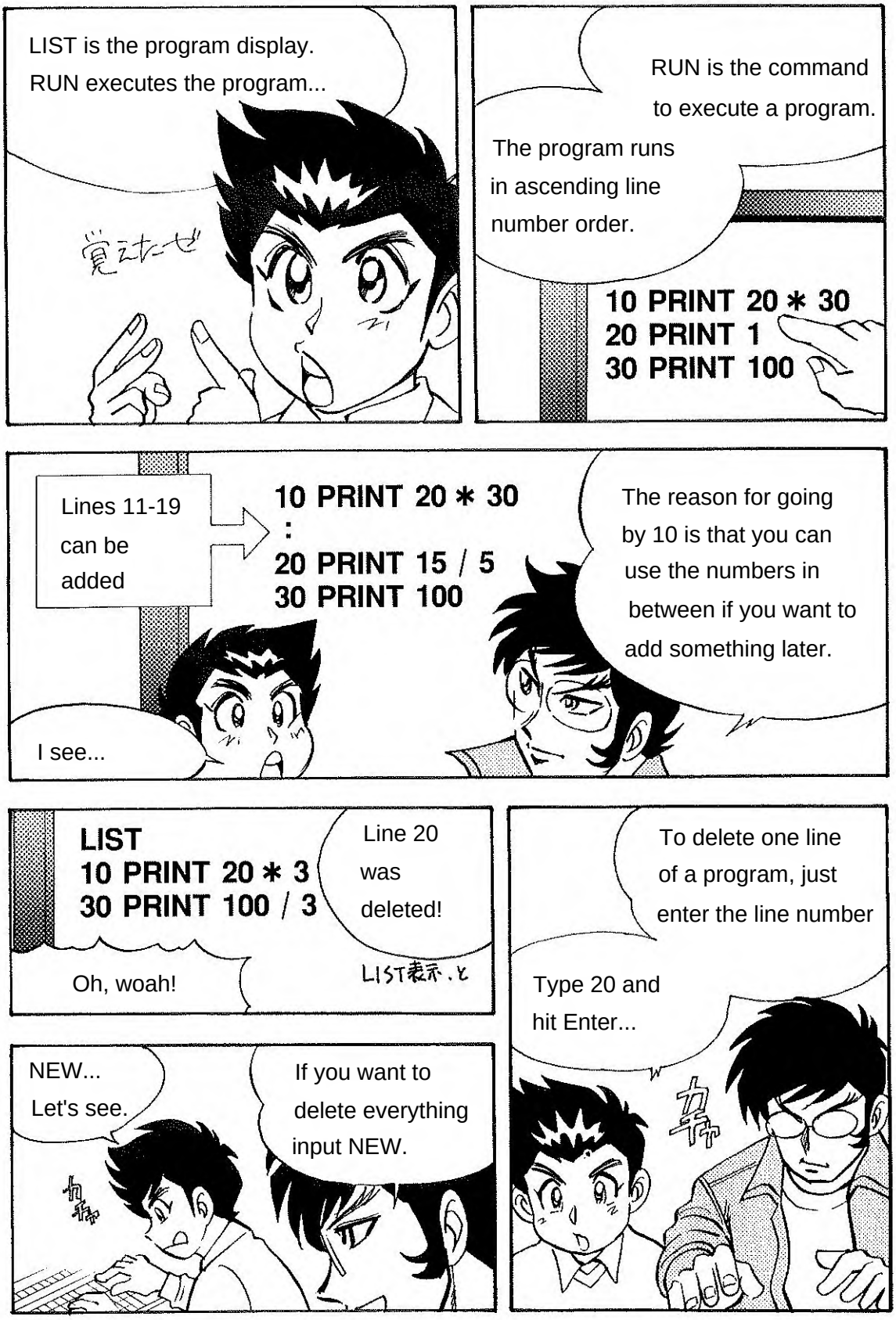

LIST
10 PRINT 20 * 30
20 PRINT 15 / 5
30 PRINT 100 / 3
■
Oh!
What I just wrote
is now displayed!

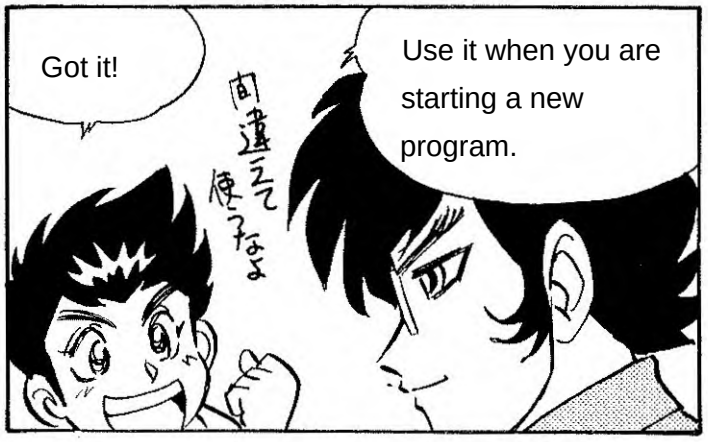
Type LIST
and see what
happens.
LIST...
Enter...

In other words,
by putting on line
numbers, the commands
are saved as a program.
LIST is a command to
display the program you
have put in so far.

RUN
実行を開始します。
600
3
33.3333
Woah!
The answers all
come out at once!

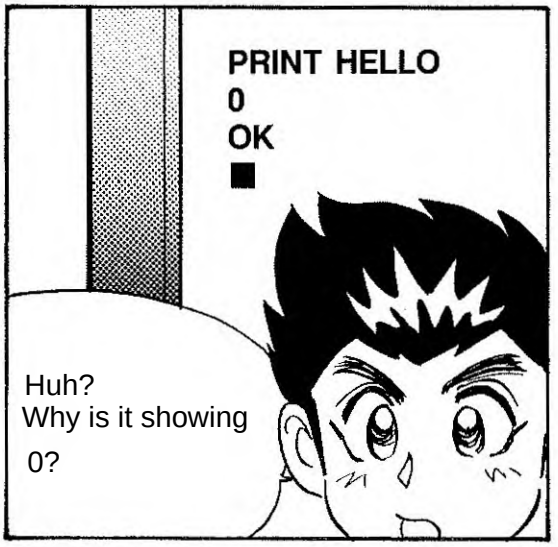

Try typing RUN
and execute.
RUN...
Enter...


LIST is the program display.
RUN executes the program...
覚えたぜ
RUN is the command to execute a program.
The program runs in ascending line number order.
10 PRINT 20 * 30
20 PRINT 1
30 PRINT 100
Lines 11-19 can be added
10 PRINT 20 * 30
:
20 PRINT 15 / 5
30 PRINT 100
The reason for going by 10 is that you can use the numbers in between if you want to add something later.
I see...
LIST
10 PRINT 20 * 3
30 PRINT 100 / 3
Line 20 was deleted!
Oh, woah!
LIST表示、と
To delete one line of a program, just enter the line number
Type 20 and hit Enter...
NEW...
Let's see.
If you want to delete everything input NEW.

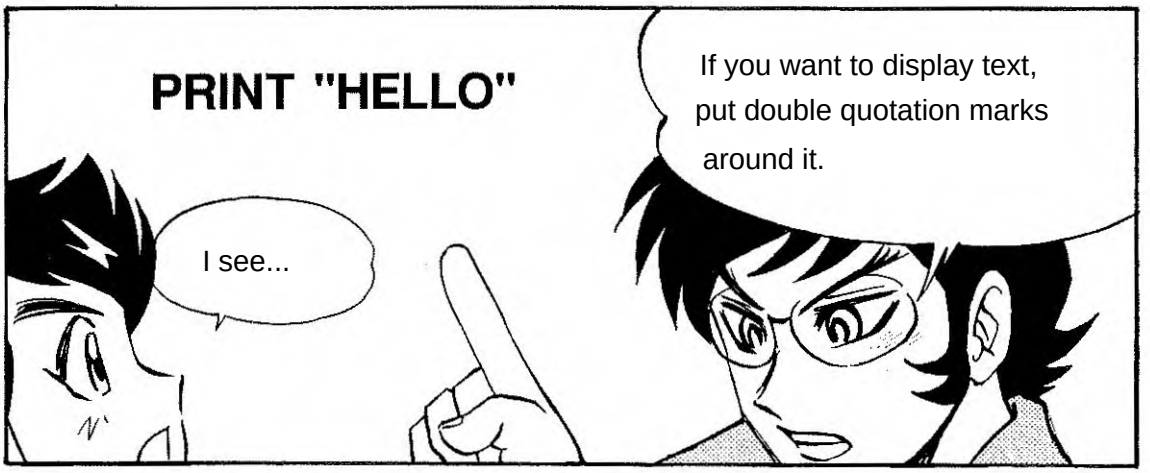

Got it!
間違えて使うなよ
Use it when you are starting a new program.

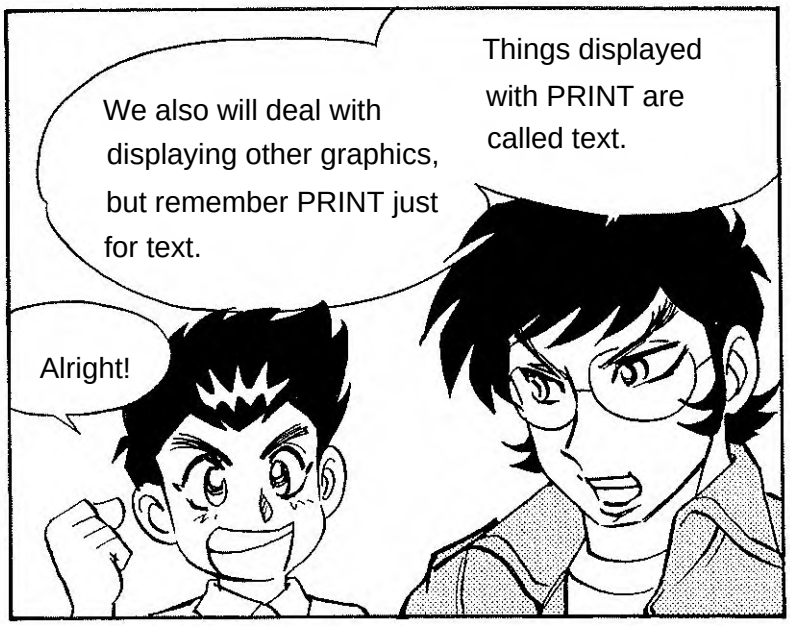
LIST
OK
■
Oh it's really all gone...

PRINT HELLO ■
Like this?
カチャカチャ

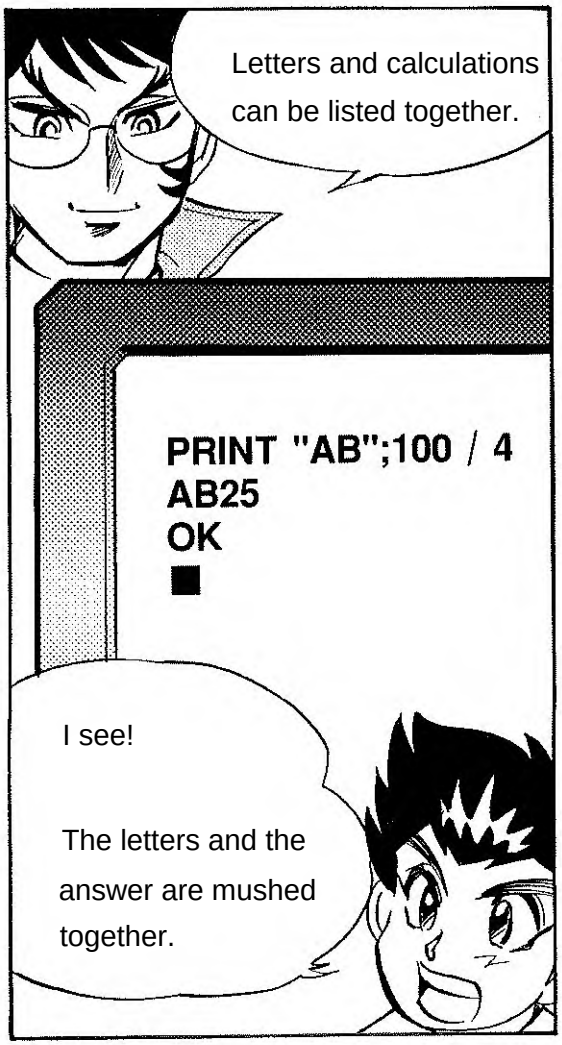

PRINT HELLO
0
OK
■
Huh?
Why is it showing 0?

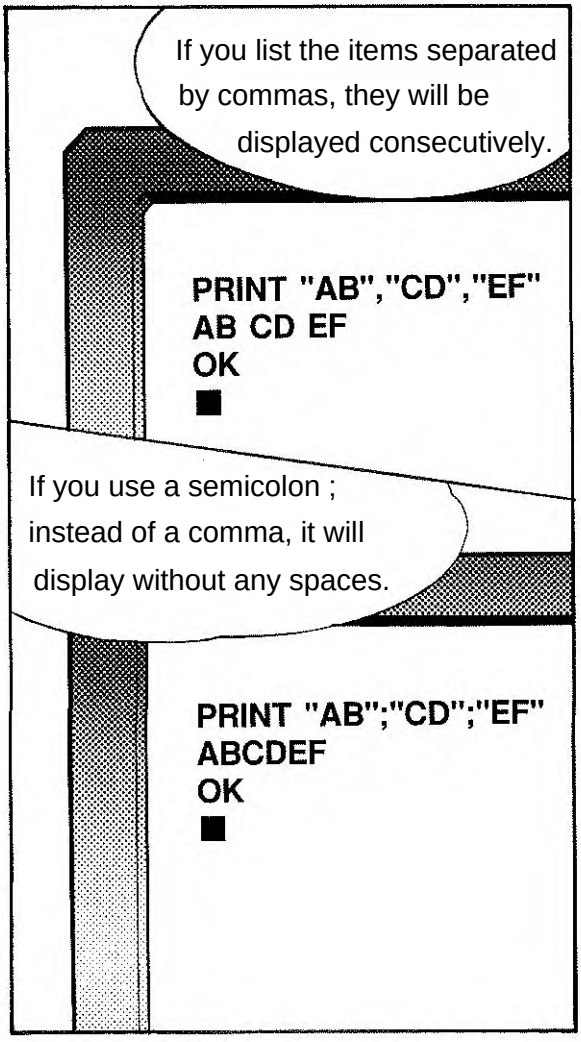
Displaying text
Next let's try displaying text.
Let's make it display HELLO.

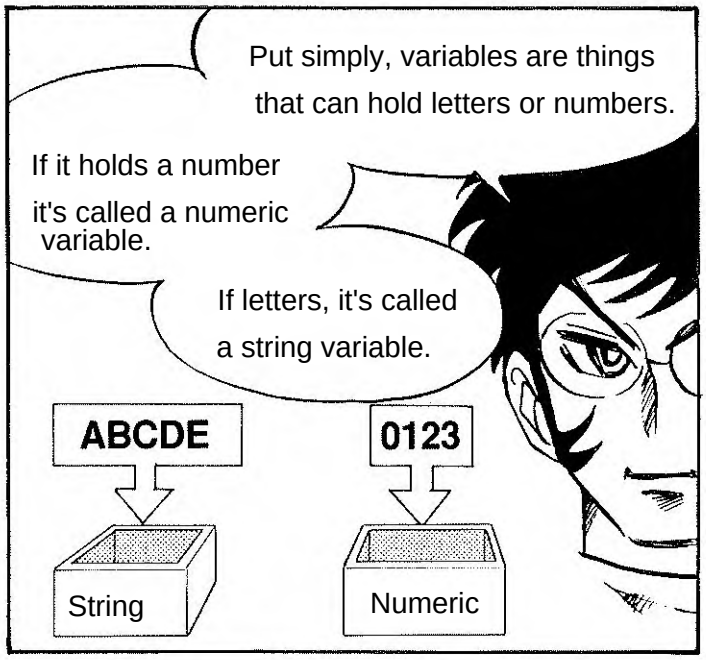

PRINT "HELLO"
I see...
If you want to display text, put double quotation marks around it.

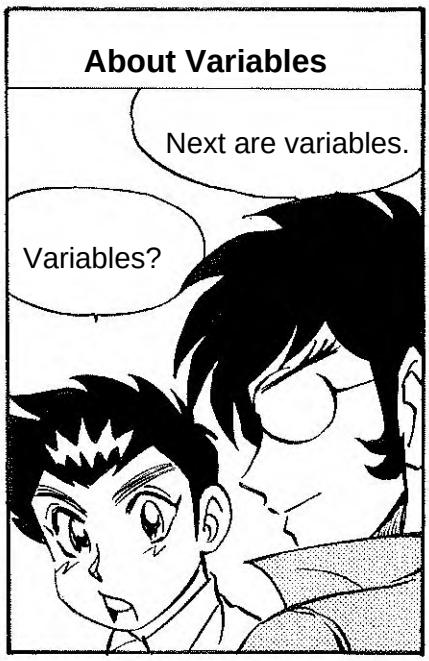

We also will deal with displaying other graphics, but remember PRINT just for text.
Things displayed with PRINT are called text.
Alright!

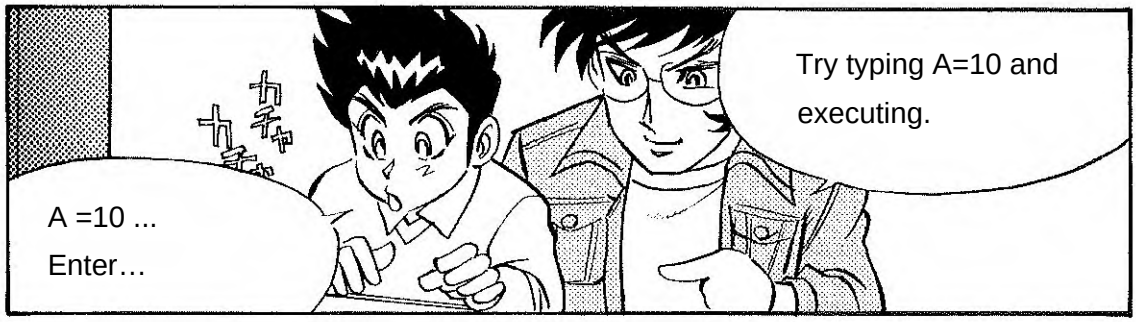
PRINT "HELLO"
HELLO
OK
■
110...
OK it worked this time!

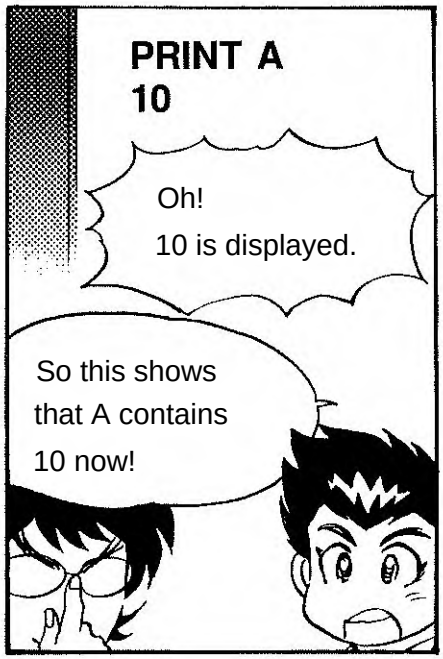

Letters and calculations can be listed together.
PRINT "AB";100 / 4
AB25
OK
■
I see!
The letters and the answer are mushed together.

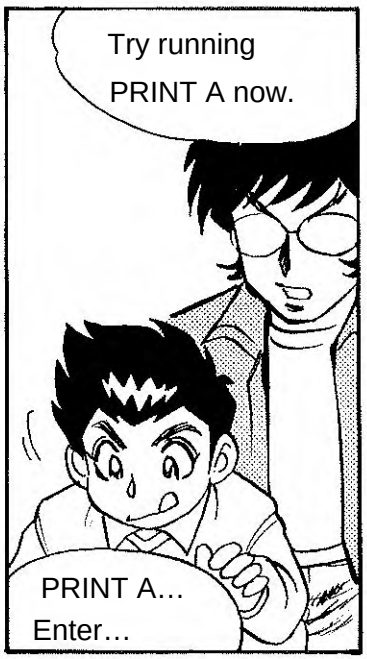

If you list the items separated by commas, they will be displayed consecutively.
PRINT "AB","CD","EF"
AB CD EF
OK
■
If you use a semicolon ; instead of a comma, it will display without any spaces.
PRINT "AB";"CD";"EF"
ABCDEF
OK
■


Put simply, variables are things that can hold letters or numbers.
If it holds a number it's called a numeric variable.
If letters, it's called a string variable.
ABCDE
0123
String
Numeric

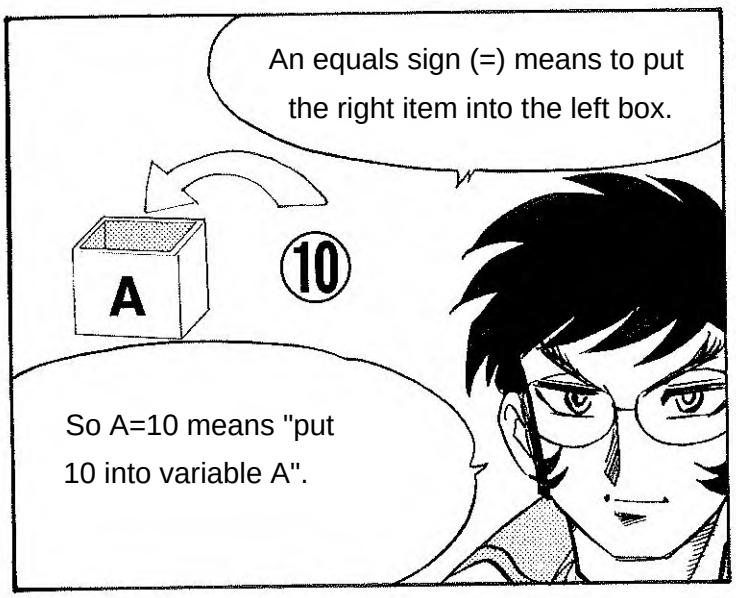

About Variables
Next are variables.
Variables?

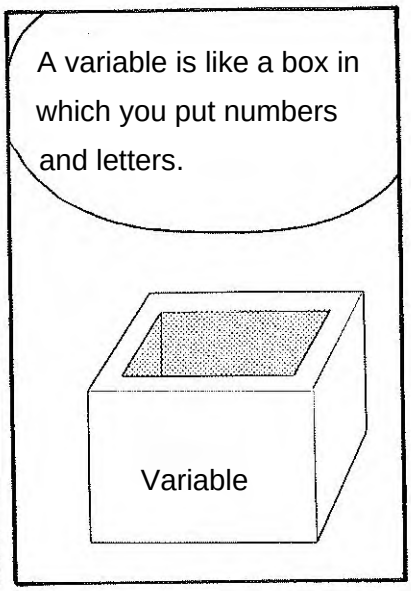

A =10 ...
Enter…
Try typing A=10 and executing.

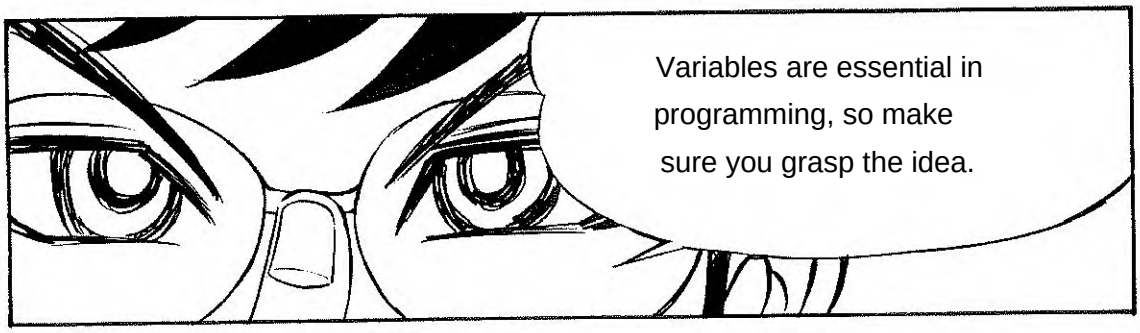

PRINT A 10
Oh!
10 is displayed.
So this shows that A contains 10 now!


Try running PRINT A now.
PRINT A…
Enter…

Nothing happened...
That line put 10 into the numeric variable A.

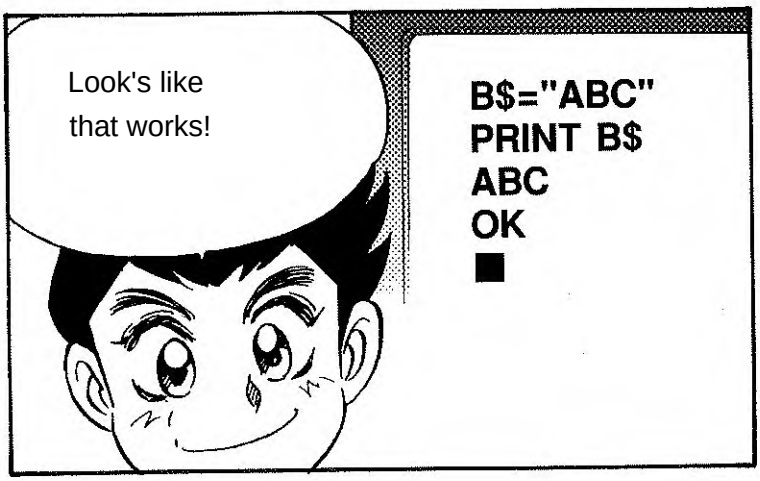

An equals sign (=) means to put the right item into the left box.
A
10
So A=10 means "put 10 into variable A".


A variable is like a box in which you put numbers and letters.
Variable


Variables are essential in programming, so make sure you grasp the idea.

PRINT to confirm...
B$="ABC"… Enter…

So what about storing letters?
Try writing B$="ABC".


Look's like that works!
B$="ABC"
PRINT B$
ABC
OK

If a variable has that $ added, it is set to be a string variable.

B$ = "ABC"

For the letters you want stored, you need to put double quotes around them.

String variables need to be named with a $ at the end.

"ABC"

The thing inside the double quotes is called a string.

○ SCORE
PT_HIT
TEXT9$

× NOTE#
51D

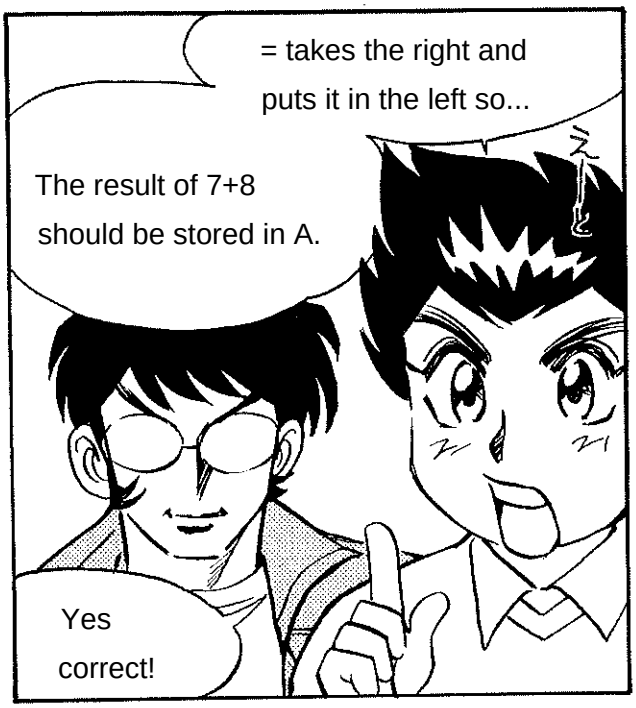
A variable name can have numbers, but the only symbol allowed is _

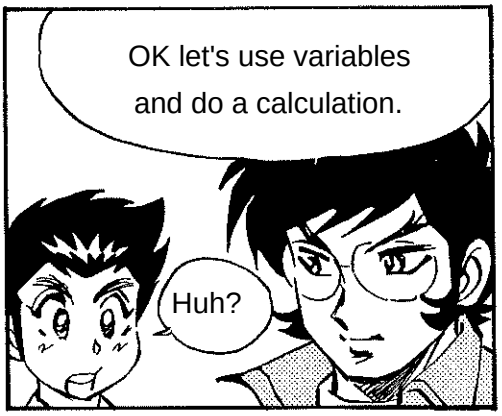
And you can't have numbers at the start.

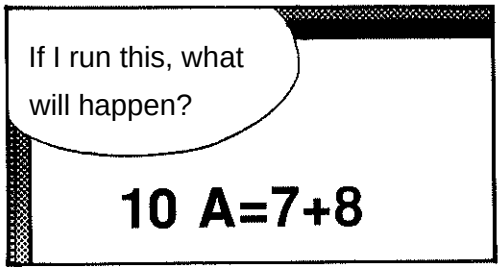
I think I get it...

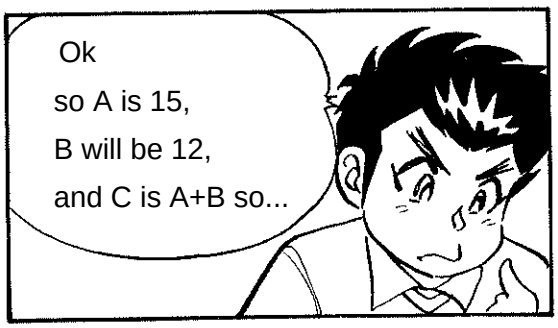
You can't put strings into numeric variables, so be careful.

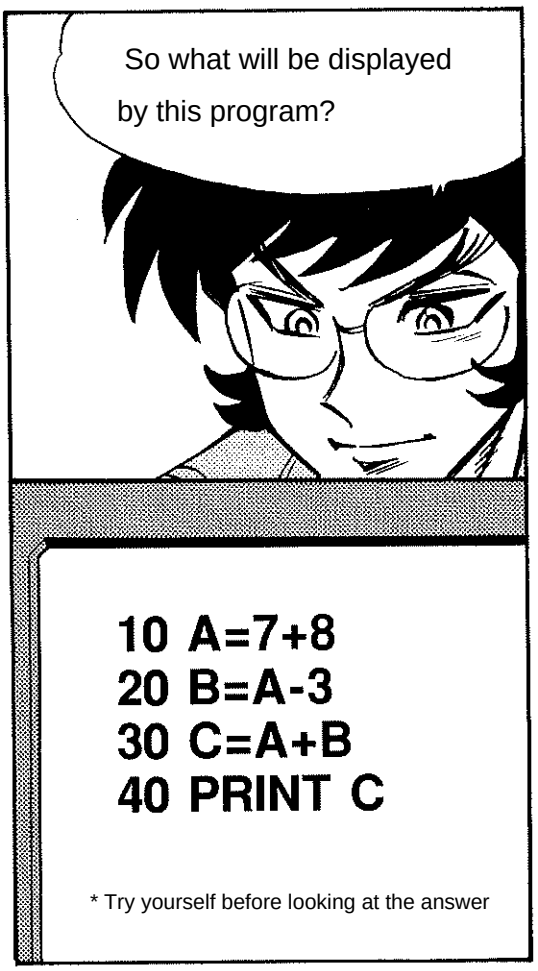
A variable name can have up to 10 characters, including the $.

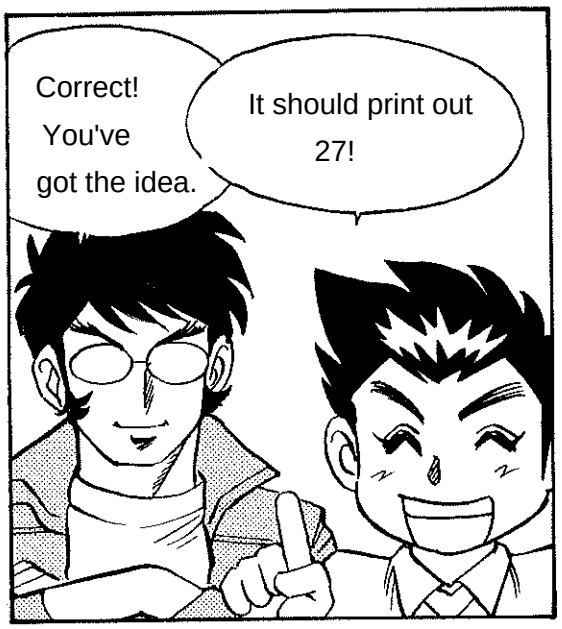

= takes the right and puts it in the left so...
The result of 7+8 should be stored in A.
Yes correct!


OK let's use variables and do a calculation.
Huh?


If I run this, what will happen?
10 A=7+8


Ok
so A is 15,
B will be 12,
and C is A+B so...


So what will be displayed by this program?
10 A=7+8
20 B=A-3
30 C=A+B
40 PRINT C
* Try yourself before looking at the answer


Correct! You've got the idea.
It should print out 27!

LIST
10 INPUT A
20 PRINT A
RUN
実行を開始します。
?
■
Huh? A question mark came up and it stopped...
OK... confirming with LIST then RUN...
Getting a value via Keyboard input
Next is storing a value in a variable with the keyboard.
10 INPUT A
20 PRINT A
Try running this program.
10 INPUT A
20 PRINT A
RUN
実行を開始します。
?
123
123
Woah!
OK I'll just do 123...
Try typing a number and hitting enter.
That's right.
INPUT is a command to take a value typed from the keyboard and assign it to a variable.
The number I typed is just displayed.

In that case, add the PROMPT command after INPUT.

After you write the string you want to display, put a : (colon) and then the variable to store the input.

```
10 INPUT PROMPT "How much?":A
RUN
実行を開始します。
How much?
■
```

I see, PROMPT dispays the text, sort of the same way that PRINT does.

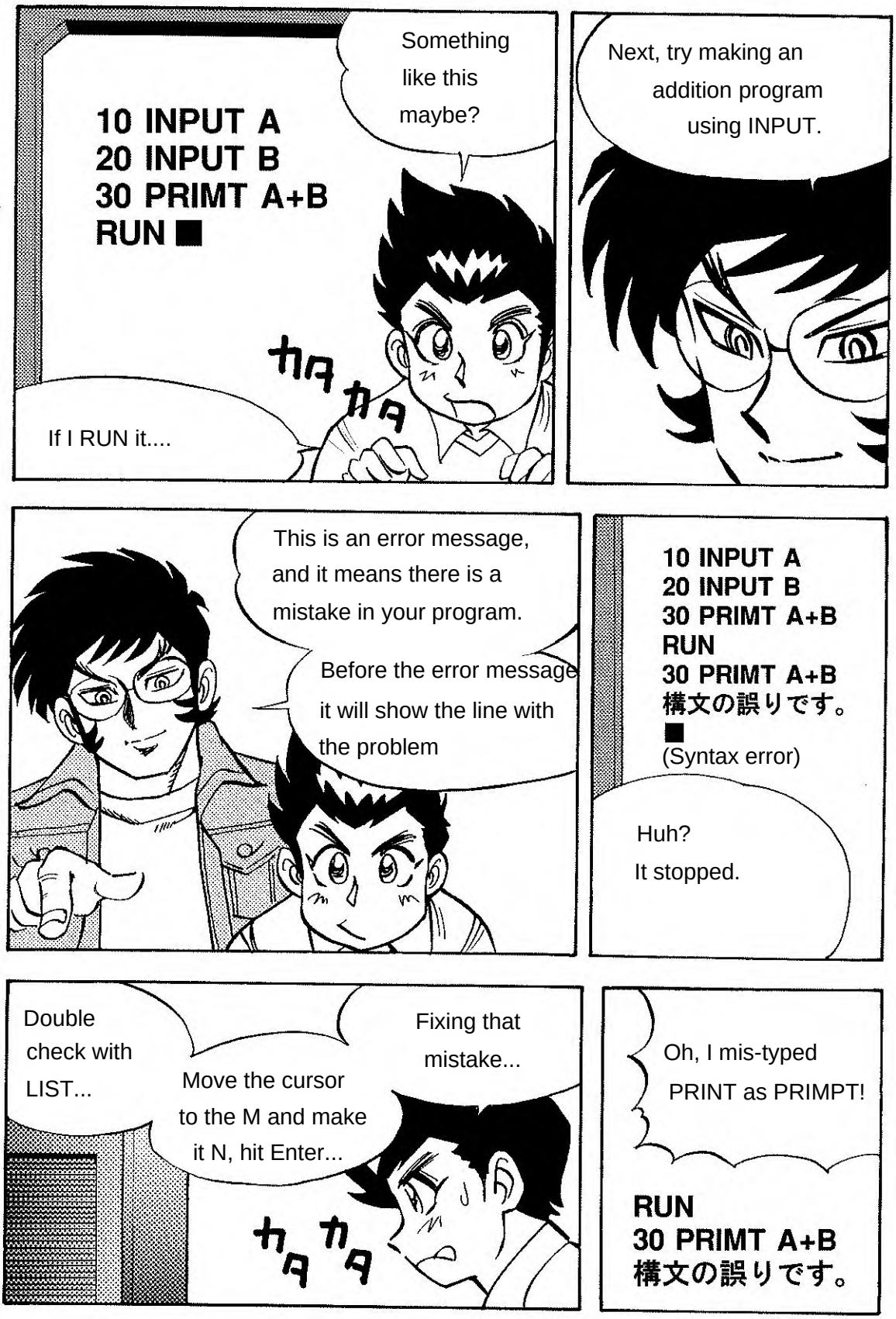
The last program puts the input value in A and displays A's value.

You can also display something other than ? when waiting for the input.

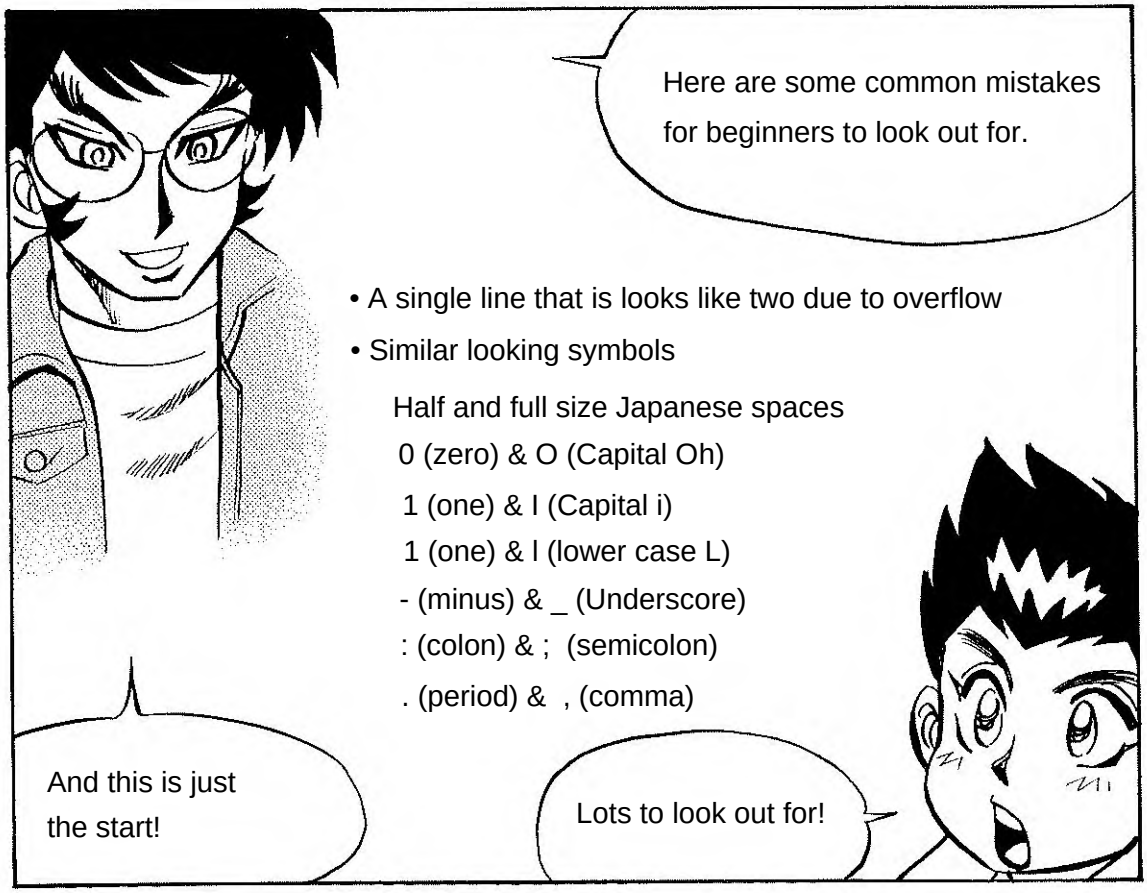
Alright!

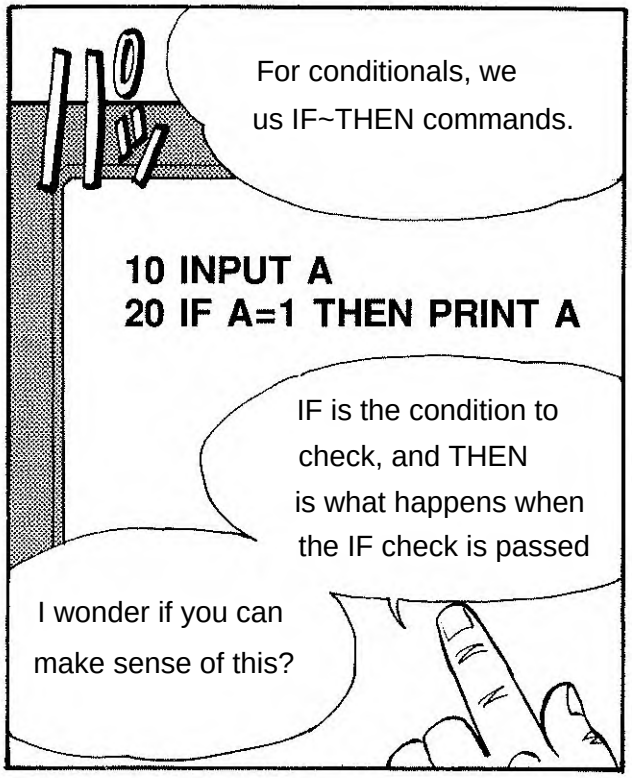
You can input to string variables as well.

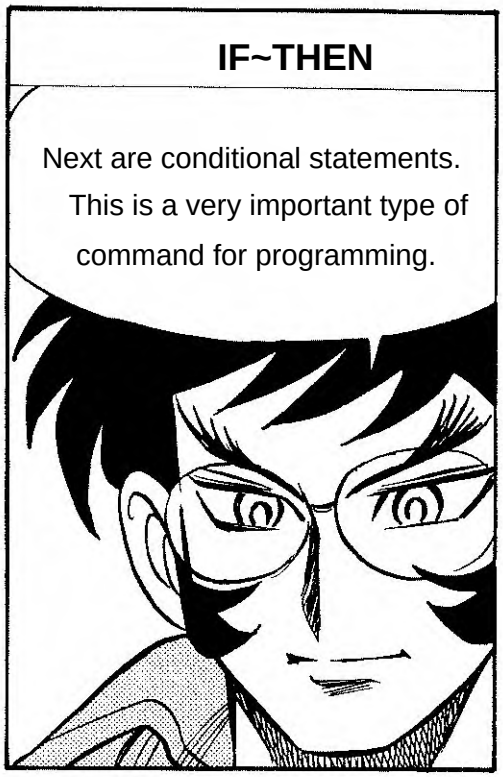
When typing in that string as input you do not need to include quotes.


10 INPUT A
20 INPUT B
30 PRIMT A+B
RUN ■
Something like this maybe?
カタカタ
If I RUN it....
Next, try making an addition program using INPUT.
This is an error message, and it means there is a mistake in your program.
Before the error message it will show the line with the problem
10 INPUT A
20 INPUT B
30 PRIMT A+B
RUN
30 PRIMT A+B
構文の誤りです。
■
(Syntax error)
Huh?
It stopped.
Double check with LIST...
Move the cursor to the M and make it N, hit Enter...
Fixing that mistake...
カタカタ
Oh, I mis-typed PRINT as PRIMPT!
RUN
30 PRIMT A+B
構文の誤りです。

Mistakes happen when writing programs!
When that happens, carefully look at the error message!
Got it!
20 INPUT B
30 PRINT A+B
RUN
実行を開始します
?
123
?
234
357
Hey it worked this time!
LIST
10 INPUT A
20 INPUT B
30 PRINT A+B
RUN ■
Ok, no more mistakes, let's RUN again...
カチッ
Ent

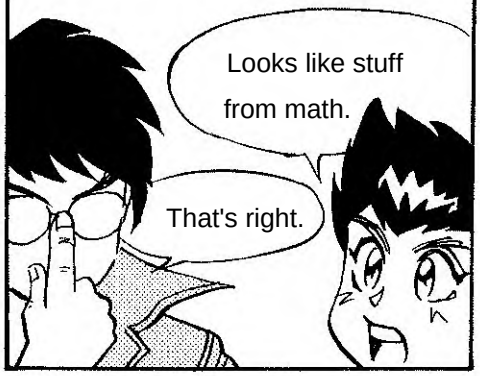

Here are some common mistakes for beginners to look out for.
• A single line that is looks like two due to overflow
• Similar looking symbols
Half and full size Japanese spaces
0 (zero) & O (Capital Oh)
1 (one) & I (Capital i)
1 (one) & l (lower case L)
- (minus) & _ (Underscore)
: (colon) & ; (semicolon)
. (period) & , (comma)
And this is just the start!
Lots to look out for!

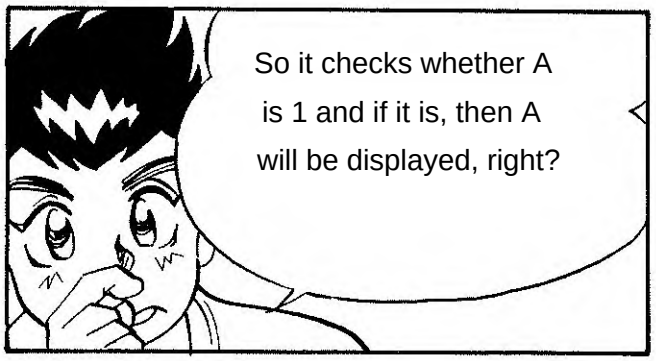

For conditionals, we us IF~THEN commands.
10 INPUT A
20 IF A=1 THEN PRINT A
IF is the condition to check, and THEN is what happens when the IF check is passed
I wonder if you can make sense of this?

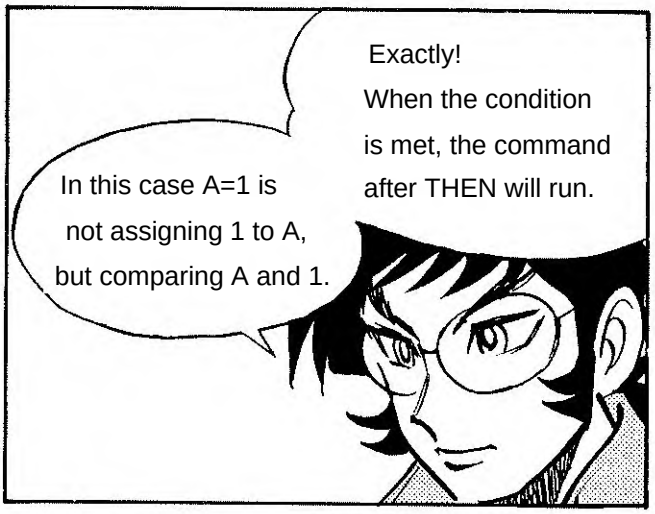


Comparison operators

= Left and right sides are same
> Left is bigger than right
< Left is smaller than right
>= Left is bigger or equal to right
<= Left is smaller or equal to right
<> Left and right are not equal






In this case A=1 is not assigning 1 to A, but comparing A and 1.
Exactly! When the condition is met, the command after THEN will run.

If the input A is bigger than 0 and B is less than 10, then it will display "OK", right?

えーと

It looks like this to have more than one condition.

```
10 INPUT A
20 INPUT B
30 IF A>0 AND B<10 THEN PRINT "OK"
```

AND checks for both conditions.
What do you think this program does?

Right!
Conditions can also be combined other ways.

AND	Both
OR	Either
NOT	if not this condition

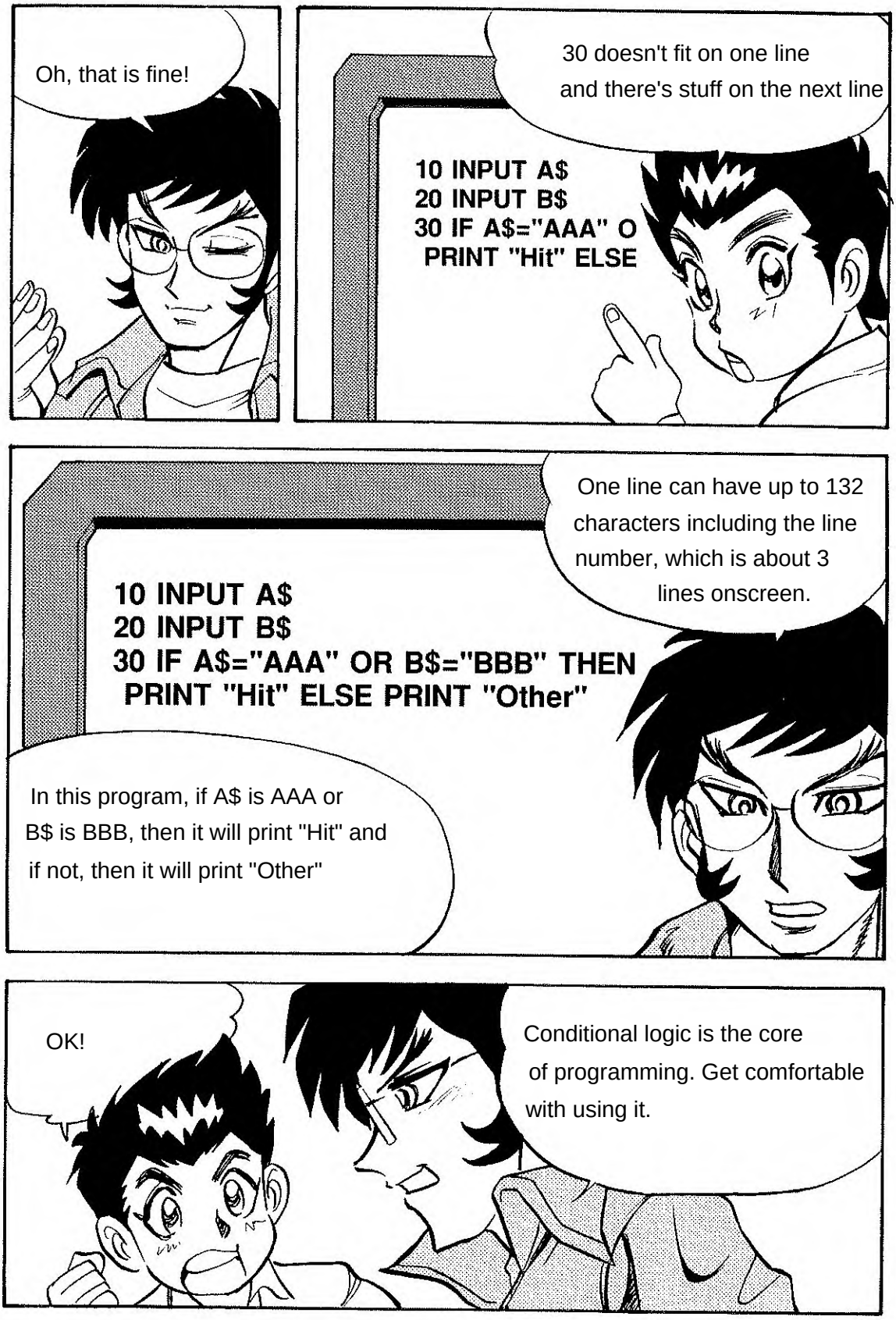
If you want to execute commands when the conditions are not met, you can add commands using ELSE.

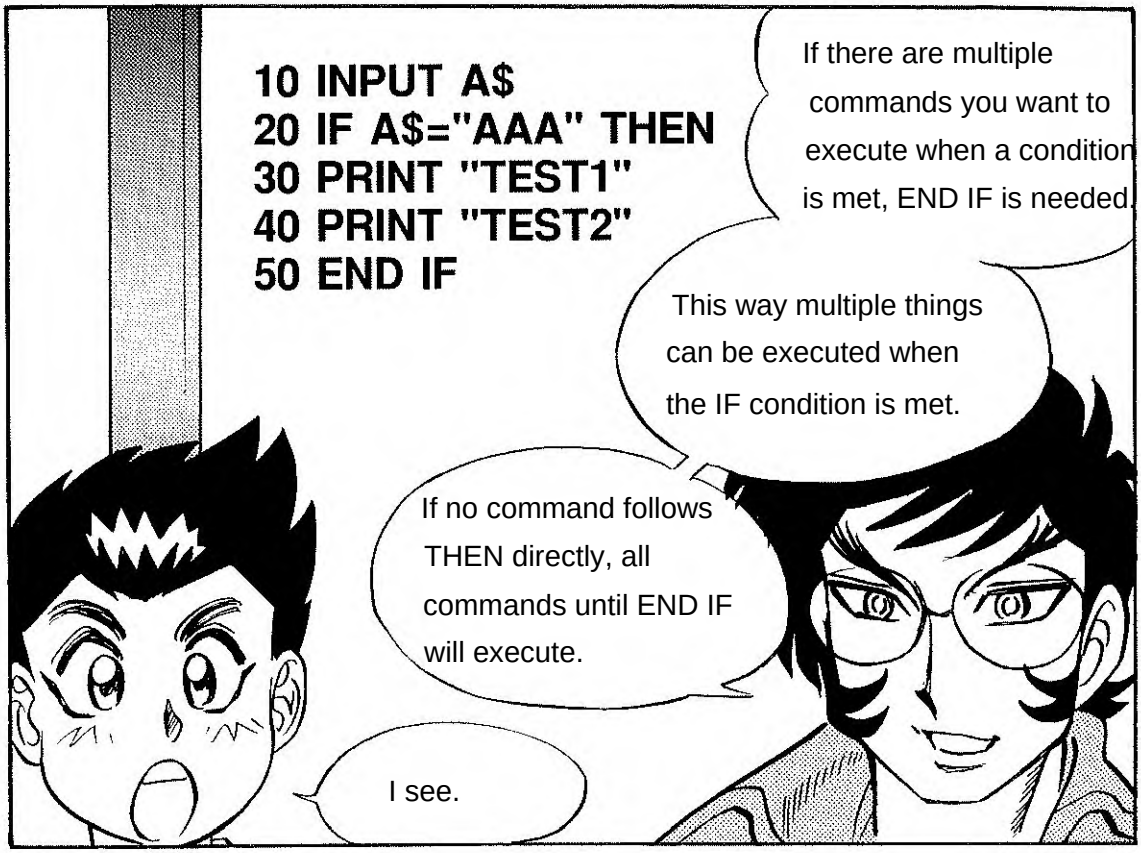
```
10 INPUT A$
20 INPUT B$
30 IF A$="AAA" OR B$="BBB" THEN
 PRINT "Hit" ELSE PRINT "Other"
```

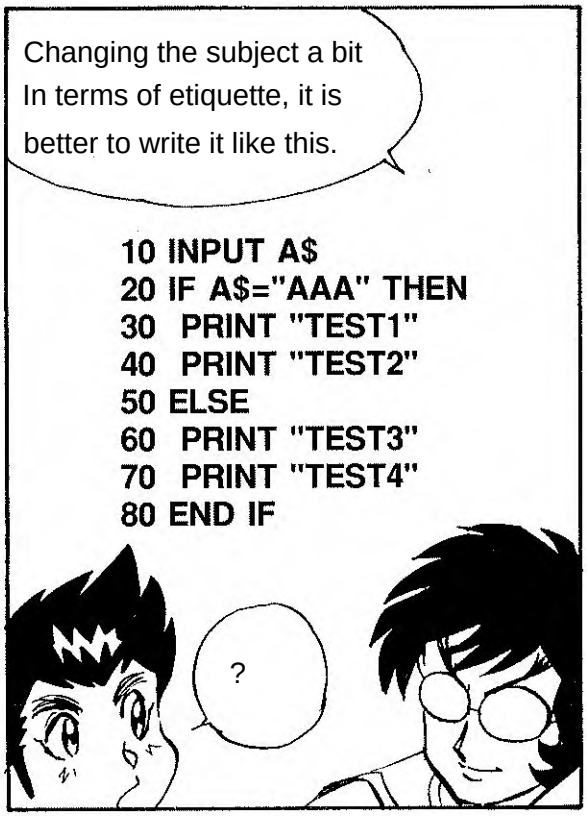
Huh?

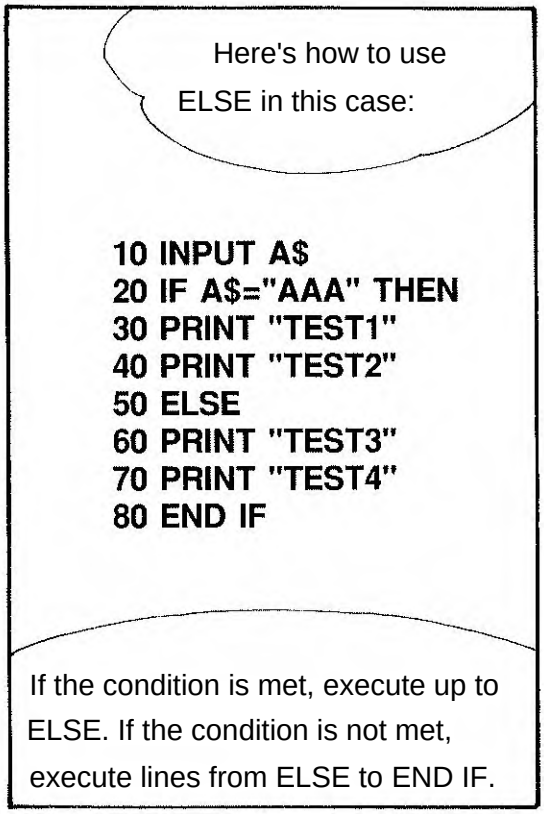
You can also compare strings.

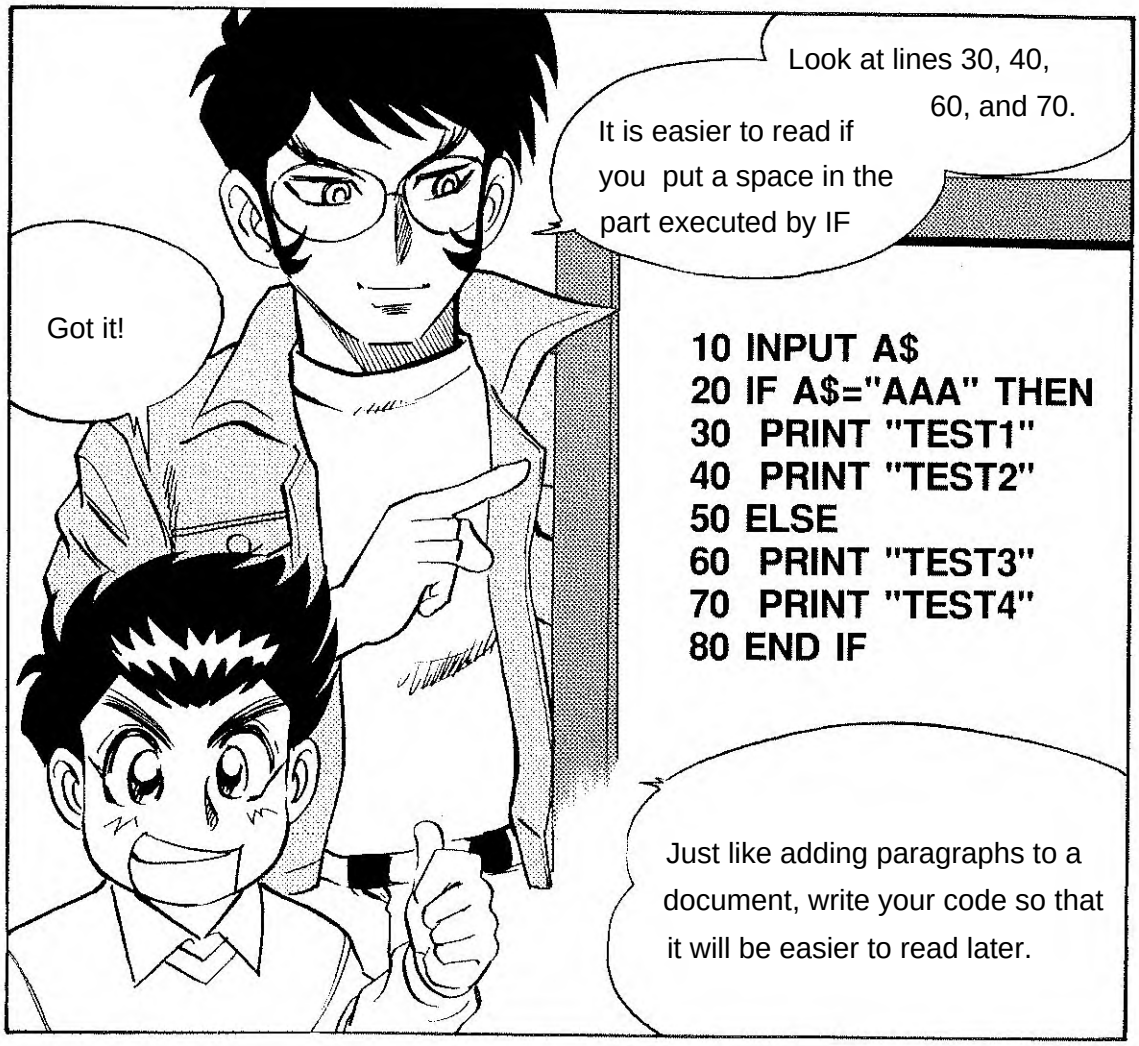
```
IF A$="AAA"
```

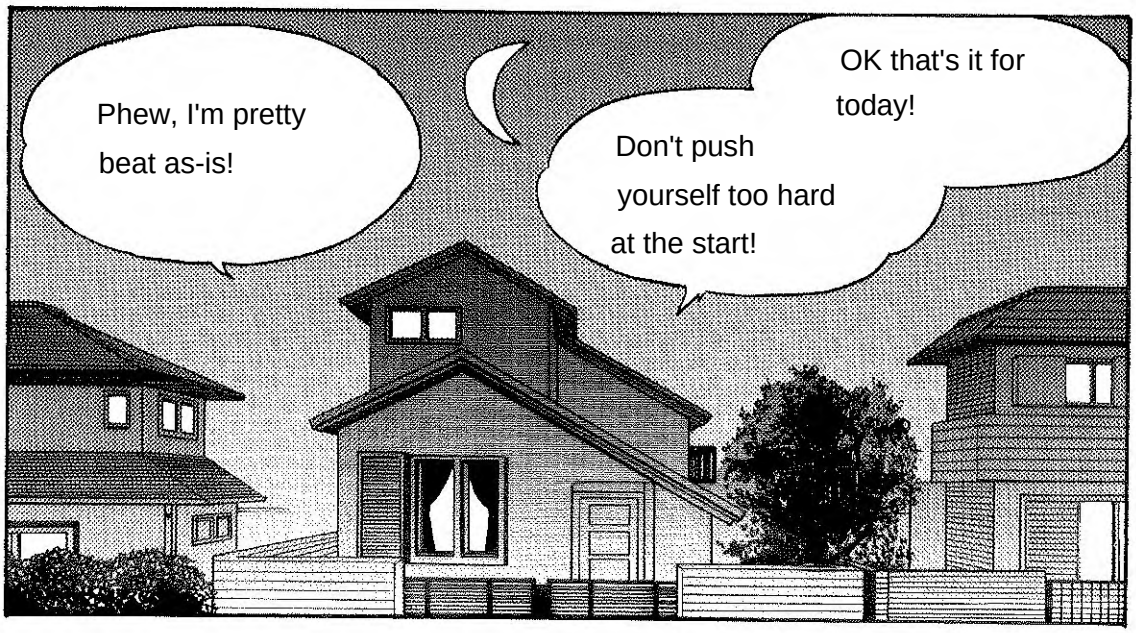

Oh, that is fine!
30 doesn't fit on one line
and there's stuff on the next line
10 INPUT A$
20 INPUT B$
30 IF A$="AAA" O
PRINT "Hit" ELSE
One line can have up to 132
characters including the line
number, which is about 3
lines onscreen.
10 INPUT A$
20 INPUT B$
30 IF A$="AAA" OR B$="BBB" THEN
PRINT "Hit" ELSE PRINT "Other"
In this program, if A$ is AAA or
B$ is BBB, then it will print "Hit" and
if not, then it will print "Other"
OK!
Conditional logic is the core
of programming. Get comfortable
with using it.


10 INPUT A$
20 IF A$="AAA" THEN
30 PRINT "TEST1"
40 PRINT "TEST2"
50 END IF
If there are multiple commands you want to execute when a condition is met, END IF is needed.
This way multiple things can be executed when the IF condition is met.
If no command follows THEN directly, all commands until END IF will execute.
I see.


Changing the subject a bit In terms of etiquette, it is better to write it like this.
10 INPUT A$
20 IF A$="AAA" THEN
30 PRINT "TEST1"
40 PRINT "TEST2"
50 ELSE
60 PRINT "TEST3"
70 PRINT "TEST4"
80 END IF
?


Here's how to use ELSE in this case:
10 INPUT A$
20 IF A$="AAA" THEN
30 PRINT "TEST1"
40 PRINT "TEST2"
50 ELSE
60 PRINT "TEST3"
70 PRINT "TEST4"
80 END IF
If the condition is met, execute up to ELSE. If the condition is not met, execute lines from ELSE to END IF.


Look at lines 30, 40, 60, and 70.
It is easier to read if you put a space in the part executed by IF
Got it!
10 INPUT A$
20 IF A$="AAA" THEN
30 PRINT "TEST1"
40 PRINT "TEST2"
50 ELSE
60 PRINT "TEST3"
70 PRINT "TEST4"
80 END IF
Just like adding paragraphs to a document, write your code so that it will be easier to read later.


Phew, I'm pretty beat as-is!
OK that's it for today!
Don't push yourself too hard at the start!

コラム1 日本語を入力してみよう

"EZ-BASIC"では、日本語を使うことができます。

Using the Software Keyboard

You can use the controller to input Japanese using the software keyboard.

♦Japanese input mode switching

♦ Pressing the "Kanji" key will switch to Japanese input mode.
*Pressing the "Kanji" key again will cancel the Japanese input mode.

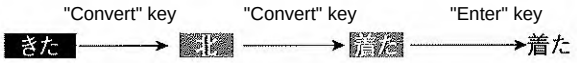
♦Enter characters

• Use the directional keys to select characters, and use the O button to enter characters. The characters you enter are displayed inverted as candidates for kanji conversion.

• Press the "Shift" key once to enter letters and symbols such as "ya", "yu", "yo", and "tsu". *Pressing the "Shift" key again will release the Shift state.

♦Kanji conversion

• Press the "Convert" key to convert the entered characters into kanji. Each time you press the "Convert" key, the next kanji candidate will be displayed.
If you press the "Shift" key, the "Convert" key becomes the "Previous Candidate" key. Pressing this to go back.
• If you are happy with the converted kanji, press the "Enter" key.



• During conversion, press the "No Conversion" key or the "ESC" key to cancel the conversion.

♦ Change input mode

• Hiragana input mode • Press the "Kana" key. You can input Hiragana and convert it to Kanji. "Akana" will be displayed in the bottom left of the screen in hiragana.
• Katakana input mode •• Press the "Kana" key while in the Shift state. Katakana can be input. "Akana" is displayed at the bottom left of the screen in katakana. Katakana input cannot be converted to Kanji.
♦ Alphabet mode Press the "English" key. You can enter full-width English characters. "A" will be displayed in the bottom left corner of the screen. Press the "English" key again to cancel.

♦Other controller operations

♦ L1 button + directional keys ----- software keyboard movement
• L2 button + directional keys - move cursor
• R1 button • … Shift key ON/OFF switch
♦ R2 button: Same function as the "Enter" key

Using the USB Keyboard

If you connect a USB keyboard to the USB port on the "PlayStation 2", you can input Japanese from the USB keyboard.
*When a USB keyboard is connected, input from the software keyboard will not be possible.

♦Japanese input mode switching

♦ Press the Half-width/Full-width key to enter Japanese input mode.
*Pressing the Half-width/Full-width key again will cancel Japanese input mode.

♦Enter characters

• Enter characters directly from the keyboard. The characters you enter are displayed as candidates for kanji conversion.

コラム1 日本語を入力してみよう

"EZ-BASIC"では、日本語を使うことができます。

When you input romaji, the alphabet displayed on the key top is input. When you input kana, the Japanese character displayed on the key top is input.

• Pressing the ESC key will erase any characters you have entered.

♦Kanji conversion

♦ Pressing the Conversion (next candidate) key will convert the entered characters into kanji.
Each time you press the Conversion (next candidate) key, the next kanji candidate will be displayed.
Each time you press the Shift key and the Conversion (next candidate) key, the previous kanji candidate will be displayed.

• If you are happy with the converted kanji, press Enter.

• Press the No Conversion key or the ESC key to cancel the conversion.

♦Switch between Kana input/Roman alphabet input

• Press the Alt key and then the Katakana/Hiragana key to switch between Kana and Romaji input.

♦ Change input mode

• Hiragana input mode: Press the Katakana/Hiragana key. You can input Hiragana and convert it to Kanji. "A Kana" or "**あ** R" will be displayed in the bottom left of the screen.

•Katakana input mode: Press the Shift key and then the Katakana/Hiragana key. You can input Katakana. "Aga"' or "**ア** R" will be displayed in the bottom left of the screen. The Katakana you input cannot be converted to Kanji.

•Alphabet mode: Press the CapLock/alphanumeric key. You can enter full-width alphanumeric characters and some symbols. “A” will be displayed in the bottom left corner of the screen.

♦Changes in conversion part

•During kanji conversion, use the left and right cursor keys to switch between words to be converted.

Right cursor key

•During kanji conversion, you can adjust the string to be converted one character at a time by pressing the Shift key and the left or right cursor key.

Shift+Left cursor key

Entering small hiragana characters

*For writing in romaji, you can do the following

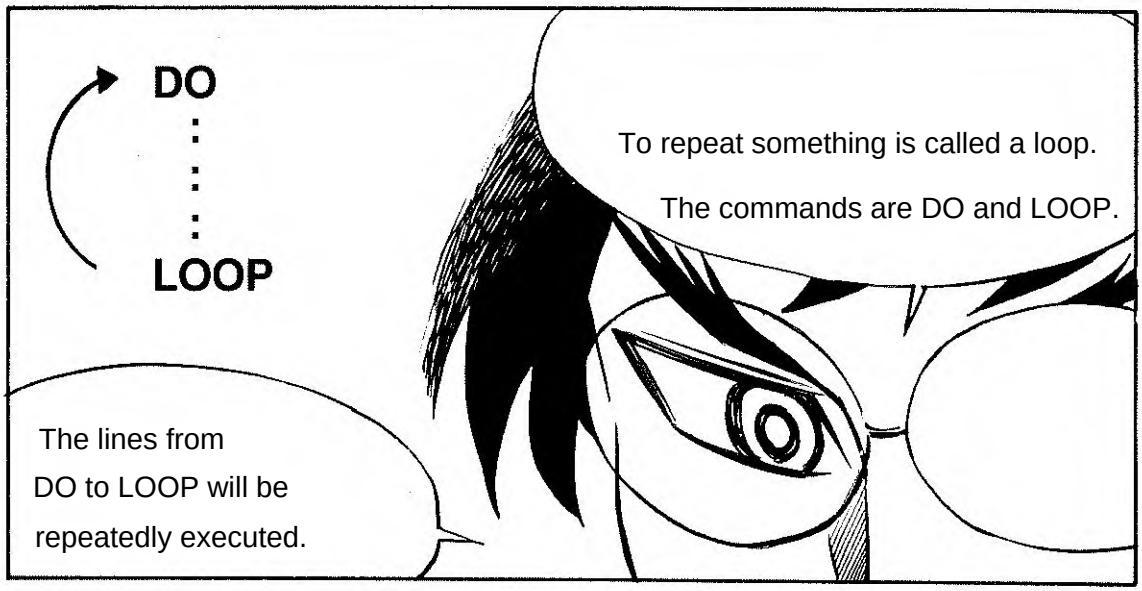
かったー	katta-
きゃんぷ	kyanpu **または** kyannpu
じえっと	jetto
しゃっくり	syakkuri

・カナ入力

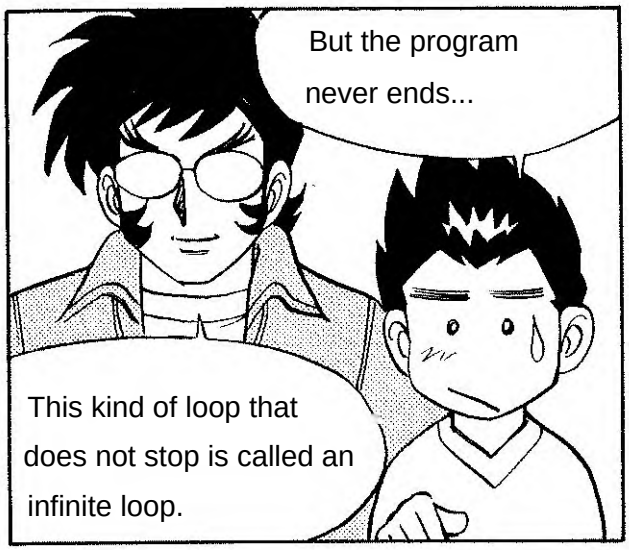
かったー	**か** Shift+**つた ー**
きゃんぷ	**き**Shift+**やんふe**
じえっと	**し，**Shift+**え** Shift+o **と**
しゃっくり	**し** Shift+**や** Shift+o **く り**

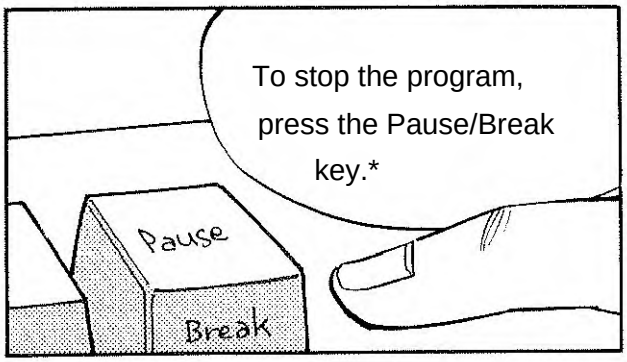
Let's make a simple game

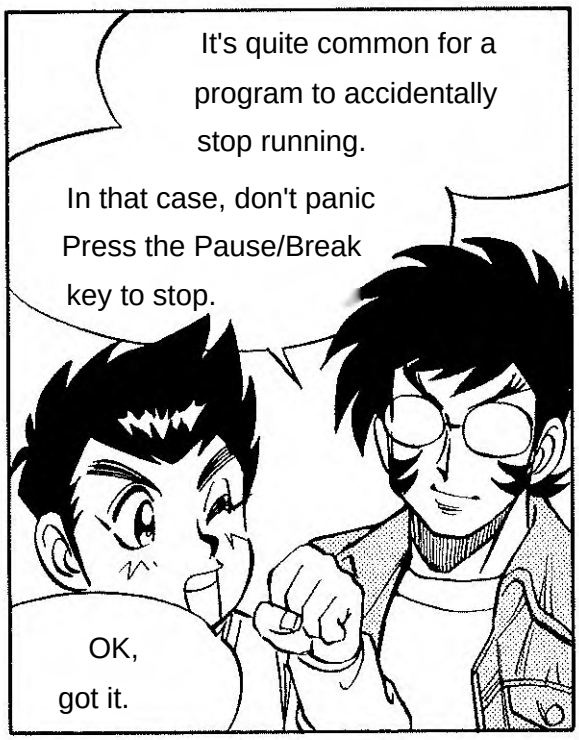
Huh ok.
Games are usually programs that run repeatedly without ending.

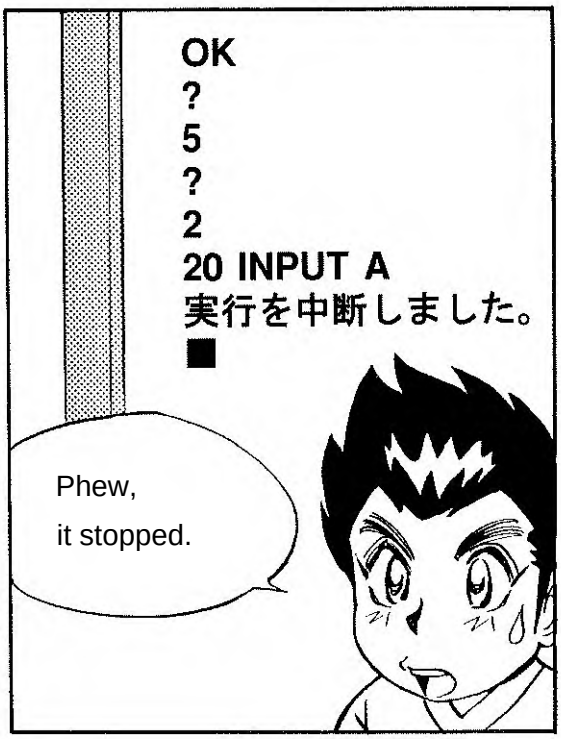

DO
LOOP
To repeat something is called a loop.
The commands are DO and LOOP.
The lines from DO to LOOP will be repeatedly executed.

10 DO
20 INPUT A
30 IF A=1 THEN PRINT "OK"
40 LOOP
Like IF, it's better to write it with the lines offset.
Specifically, it looks like this.
ha ha
Execute with RUN...






It's quite common for a program to accidentally stop running.
In that case, don't panic Press the Pause/Break key to stop.
OK, got it.



*You can also interrupt execution by pressing the START + SELECT buttons on the DUALSHOCK2 analog controller.

Here's an example.
10 DO
20 INPUT A
30 IF A=1 THEN PRINT
40 LOOP UNTIL A>10
That happened because we had no end condition for our loop in the last example.
The DO-LOOP command can have a condition for ending the loop.
By adding UNTIL after DO or LOOP, you can write the end condition.
In this example
If A exceeds 10, the execution ends without going back to DO.
I see…
Can I use AND or OR to make conditional judgments like IF?
Good question!
It works exactly the same way.

```
10 DO
20  INPUT A
30  IF A=1 THEN PRINT "OK"
40 LOOP WHILE A<=10
```

This performs the same action as the previous program with UNTIL.

Notice how the last comparison has changed slightly.

Similar to UNTIL, there is also the loop command WHILE.

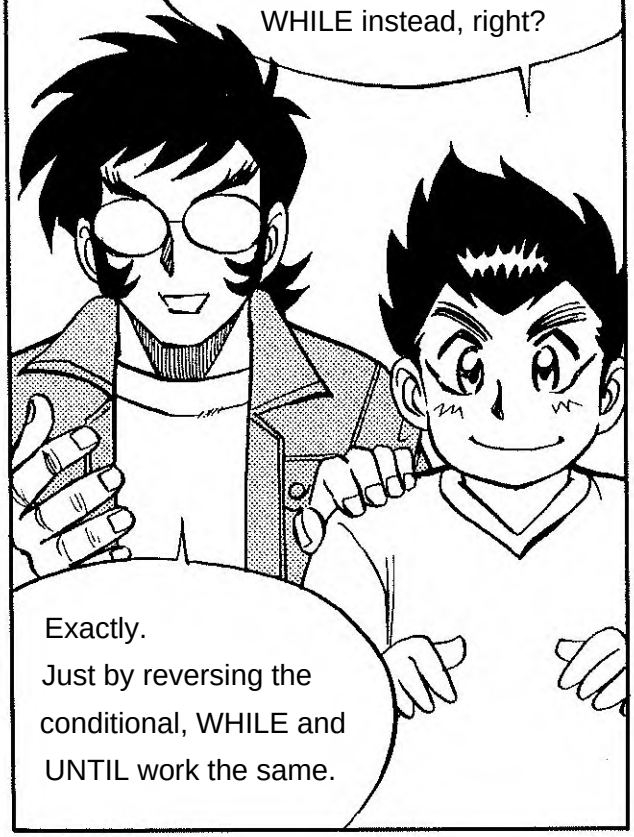
This is used with a condition for repeating, rather than a condition for ending.

How about this one?

```
10 A=50
20 DO WHILE A<=10
30  INPUT A
40  IF A=1 THEN PR
50 LOOP
```

* Try answering yourself before looking

So you basically have to reverse the condition to use WHILE instead, right?



Exactly.
Just by reversing the conditional, WHILE and UNTIL work the same.

But A is already 50...
It's not below 10, so what will happen here?

```
10 A=50
20 DO WHILE A
30  INPUT A
40  IF A=1 TH
```

Ok... put 50 in A, then....

If A is less than ten, repeat from the DO to the LOOP statement....

Hmm?

DO-LOOPs form the basis of lots of programs, so make sure to understand carefully.

In this case, nothing in the loop will ever execute.

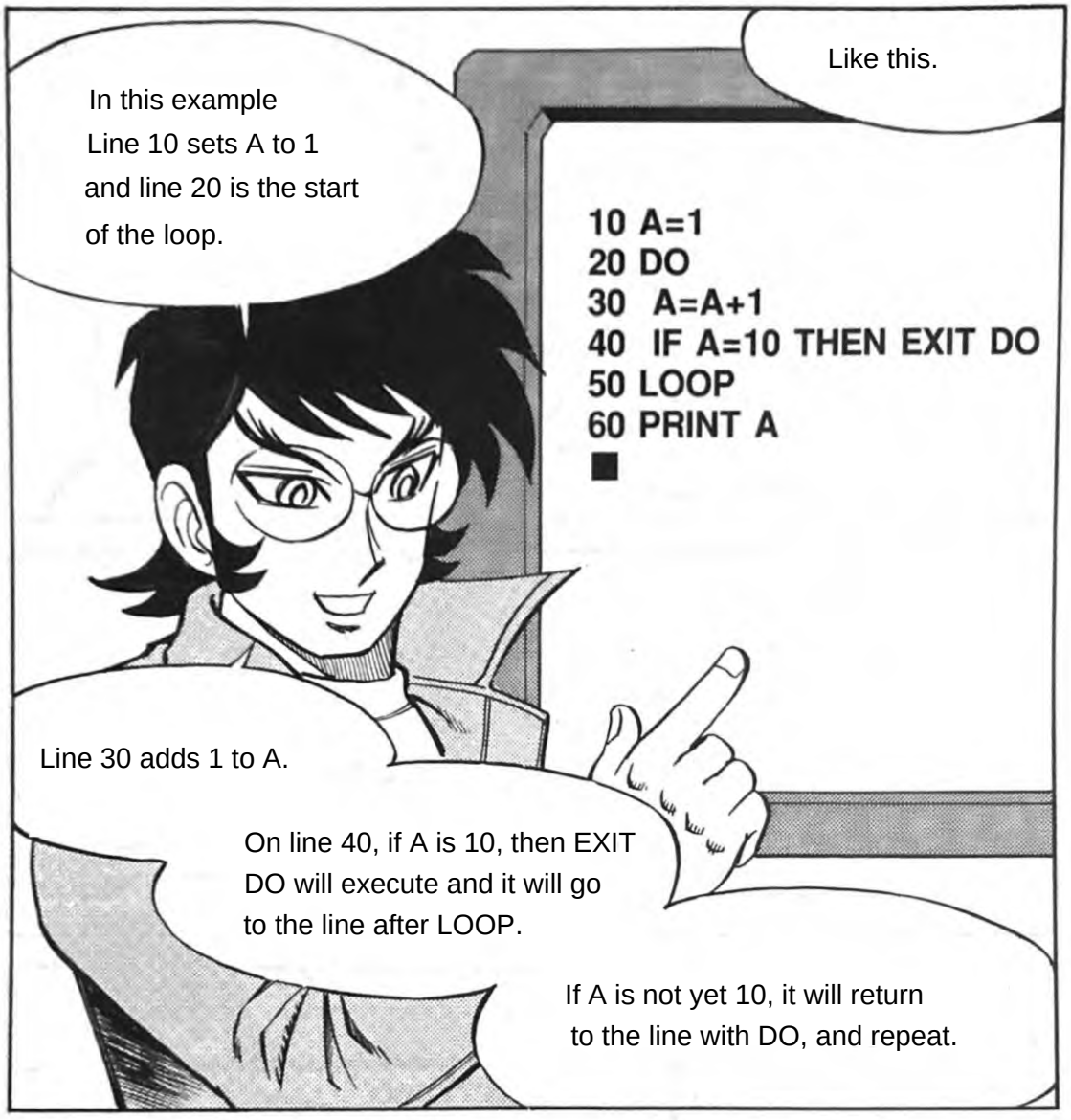
It will skip past lines 30 and 40 without executing either of them.

```
10 A=50
20 DO WHILE A<=10
30  INPUT A
40  IF A=1 THEN PR
50 LOOP
```

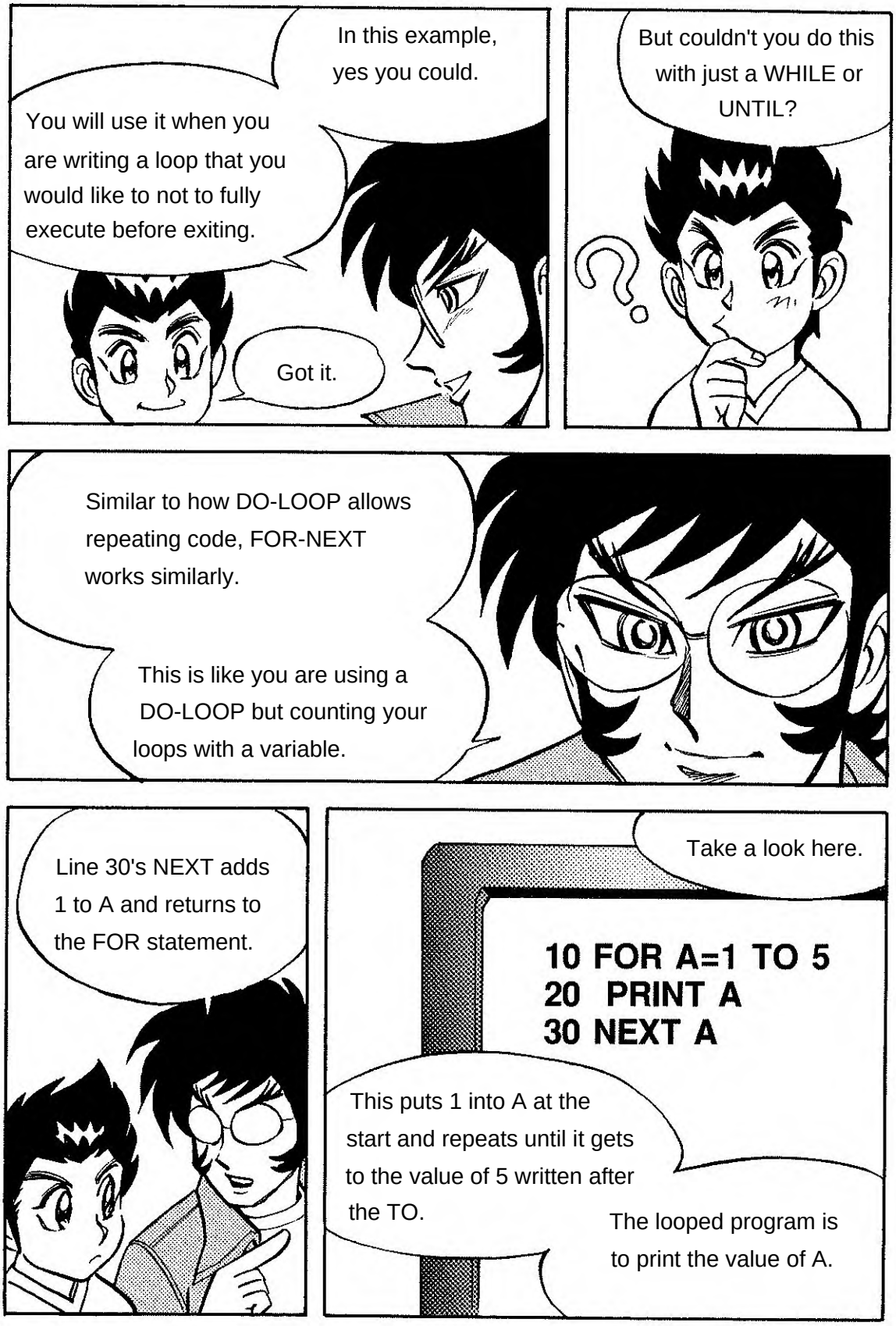
Let's teach you how to exit a loop while in the middle of it.

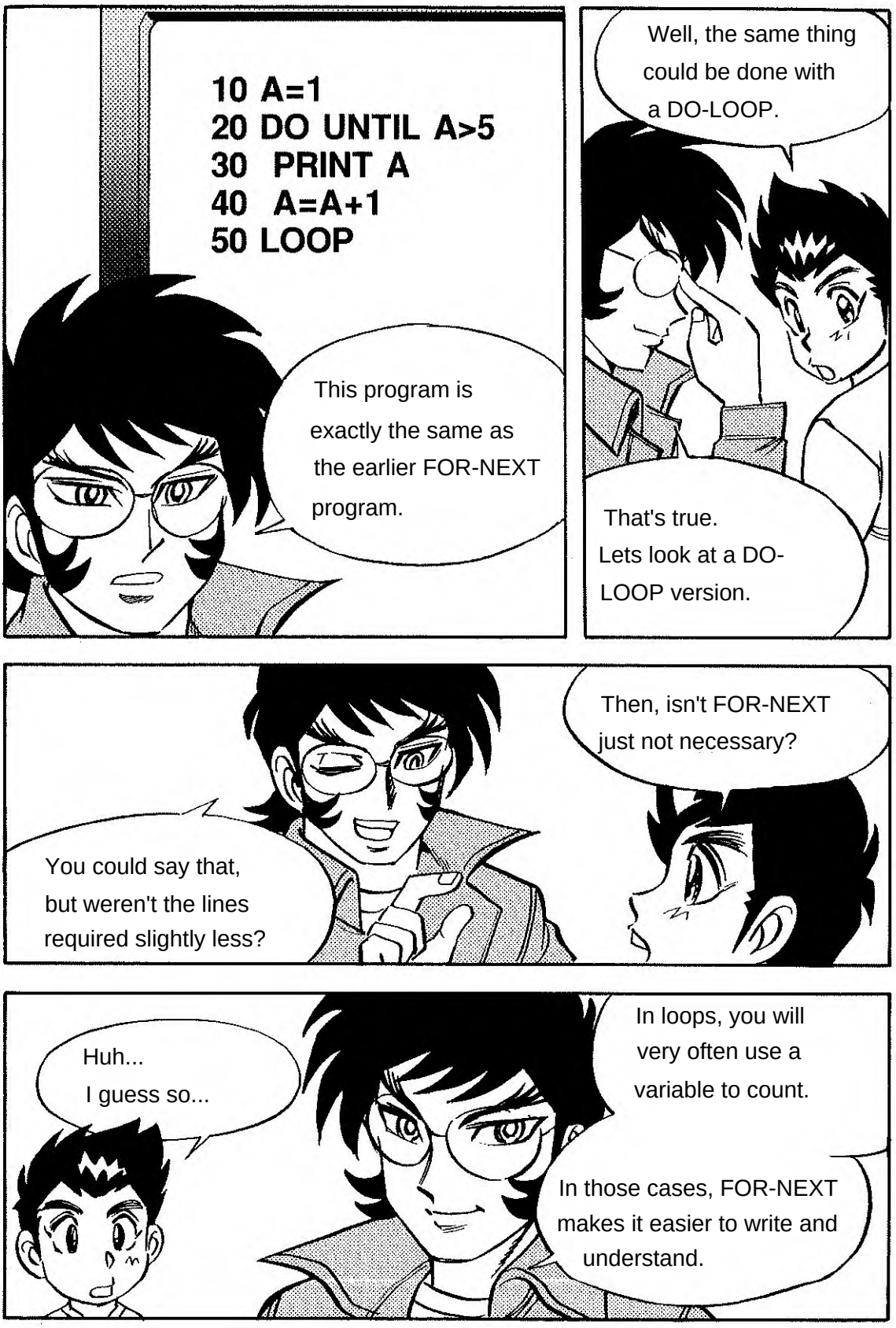
You use EXIT DO - it will jump to the end.

Huh...
I see!


Like this.
In this example
Line 10 sets A to 1
and line 20 is the start
of the loop.
10 A=1
20 DO
30 A=A+1
40 IF A=10 THEN EXIT DO
50 LOOP
60 PRINT A
Line 30 adds 1 to A.
On line 40, if A is 10, then EXIT
DO will execute and it will go
to the line after LOOP.
If A is not yet 10, it will return
to the line with DO, and repeat.

I see!
So the result after looping
will be that at line 60 A will
be displayed as 10.


In this example, yes you could.
You will use it when you are writing a loop that you would like to not to fully execute before exiting.
Got it.
But couldn't you do this with just a WHILE or UNTIL?
Similar to how DO-LOOP allows repeating code, FOR-NEXT works similarly.
This is like you are using a DO-LOOP but counting your loops with a variable.
Line 30's NEXT adds 1 to A and returns to the FOR statement.
Take a look here.
10 FOR A=1 TO 5
20 PRINT A
30 NEXT A
This puts 1 into A at the start and repeats until it gets to the value of 5 written after the TO.
The looped program is to print the value of A.


10 A=1
20 DO UNTIL A>5
30 PRINT A
40 A=A+1
50 LOOP
This program is exactly the same as the earlier FOR-NEXT program.
Well, the same thing could be done with a DO-LOOP.
That's true. Lets look at a DO-LOOP version.
Then, isn't FOR-NEXT just not necessary?
You could say that, but weren't the lines required slightly less?
Huh... I guess so...
In loops, you will very often use a variable to count.
In those cases, FOR-NEXT makes it easier to write and understand.

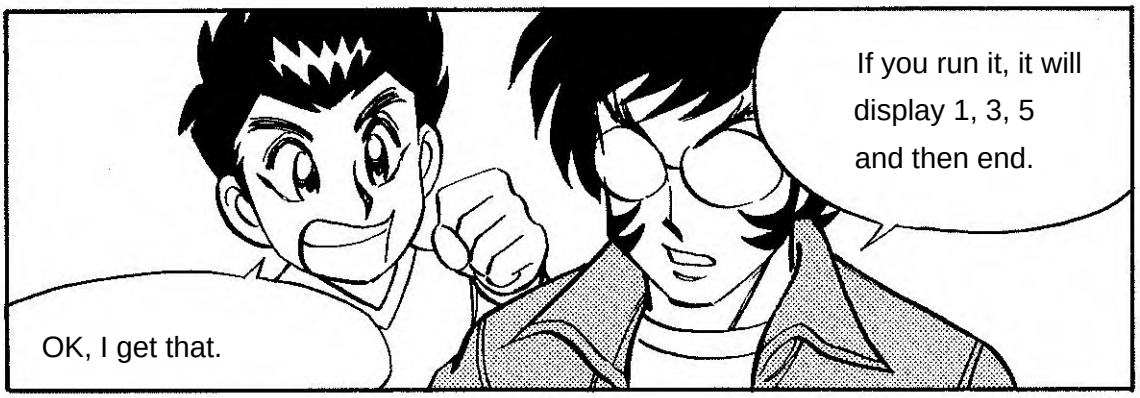
NEXT A will by default add 1 to A and loop, but you can change that amount using STEP.

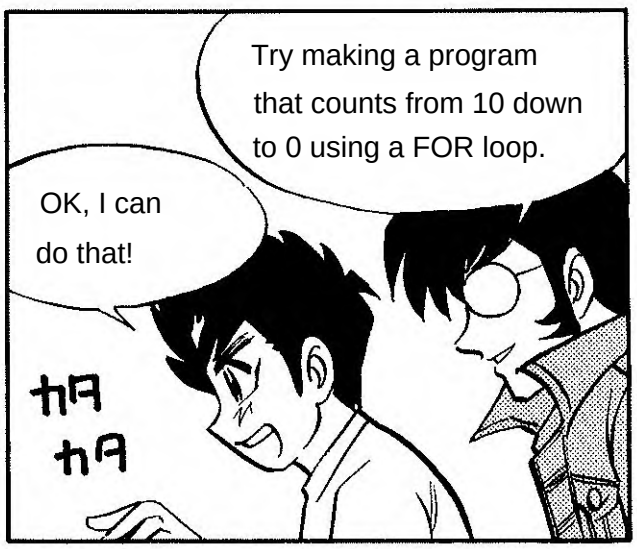
```
10 FOR A=1 TO 5 STEP 2
20  PRINT A
30 NEXT A
```

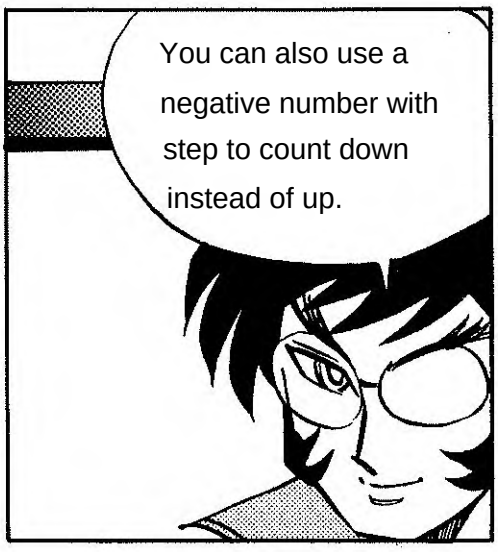
In this example, NEXT will always add 2 to A when it loops.

FOR-NEXT

So let's thoroughly go through FOR-NEXT.







That's right.
This should do it for FOR-NEXT, then.
10 FOR A=10 TO 0 STEP -1
20 PRINT A
30 NEXT A
Got it!
10 FOR A=1 TO 10
20 IF A=5 THEN EXIT FOR
30 NEXT A
40 PRINT A
Like EXIT DO, there is a command for exiting a FOR loop.
Reasonably, it's EXIT FOR.
Got it!
Just like EXIT-DO, you use it to exit a FOR-NEXT loop early.
Keep that in mind.

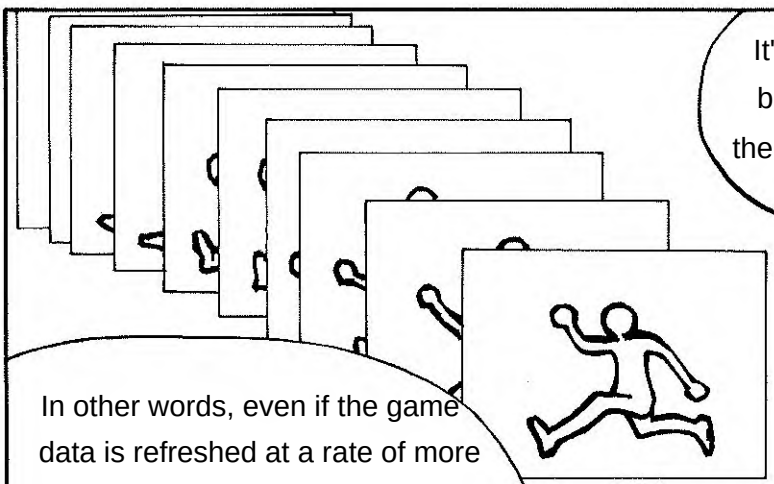
When you run a program, the display may be too fast to see.

For example, the countdown program was too fast isn't it? There's no countdown, right?

Let's do some simple commands for a break.

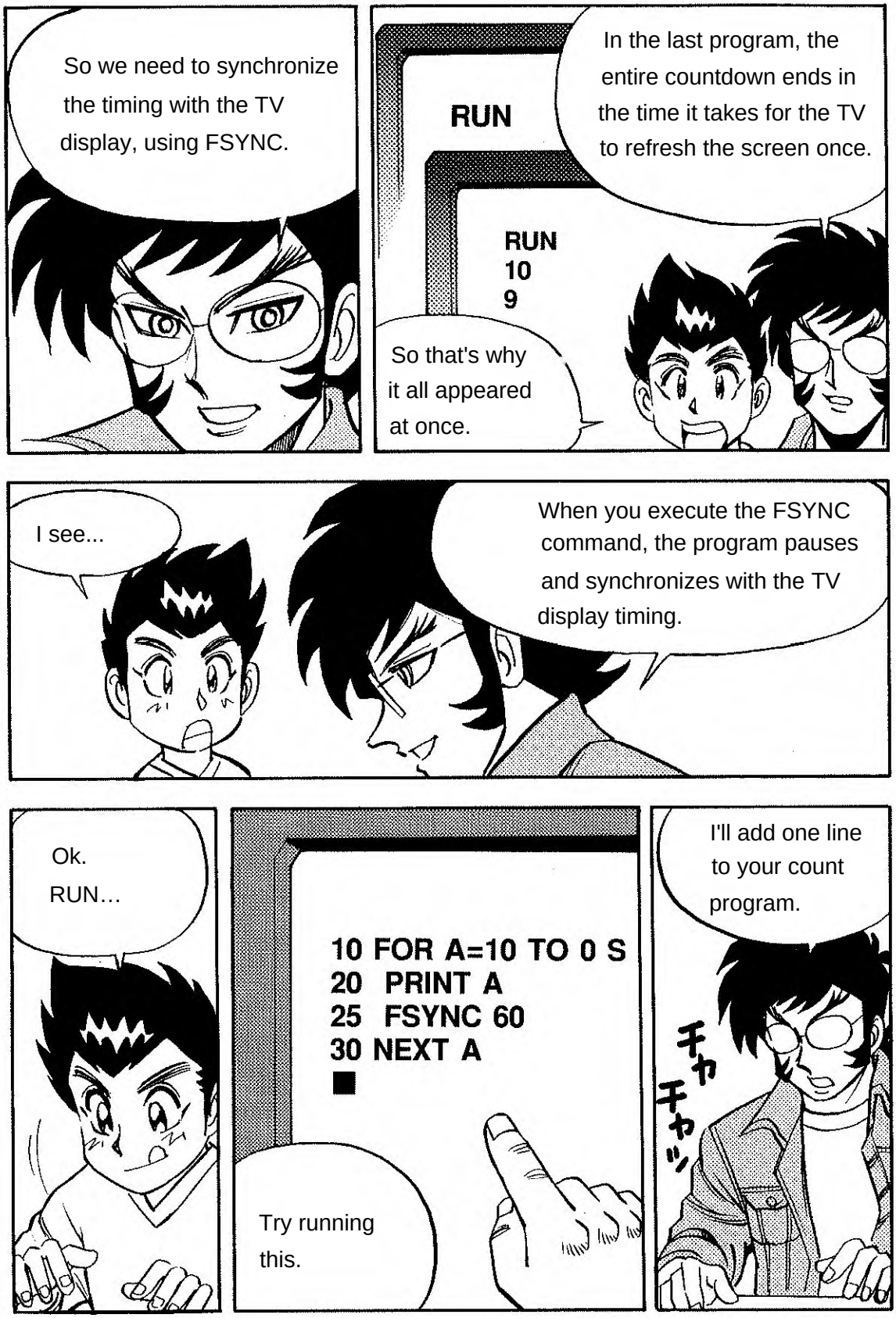
RUN
10
9
8
7
6
5

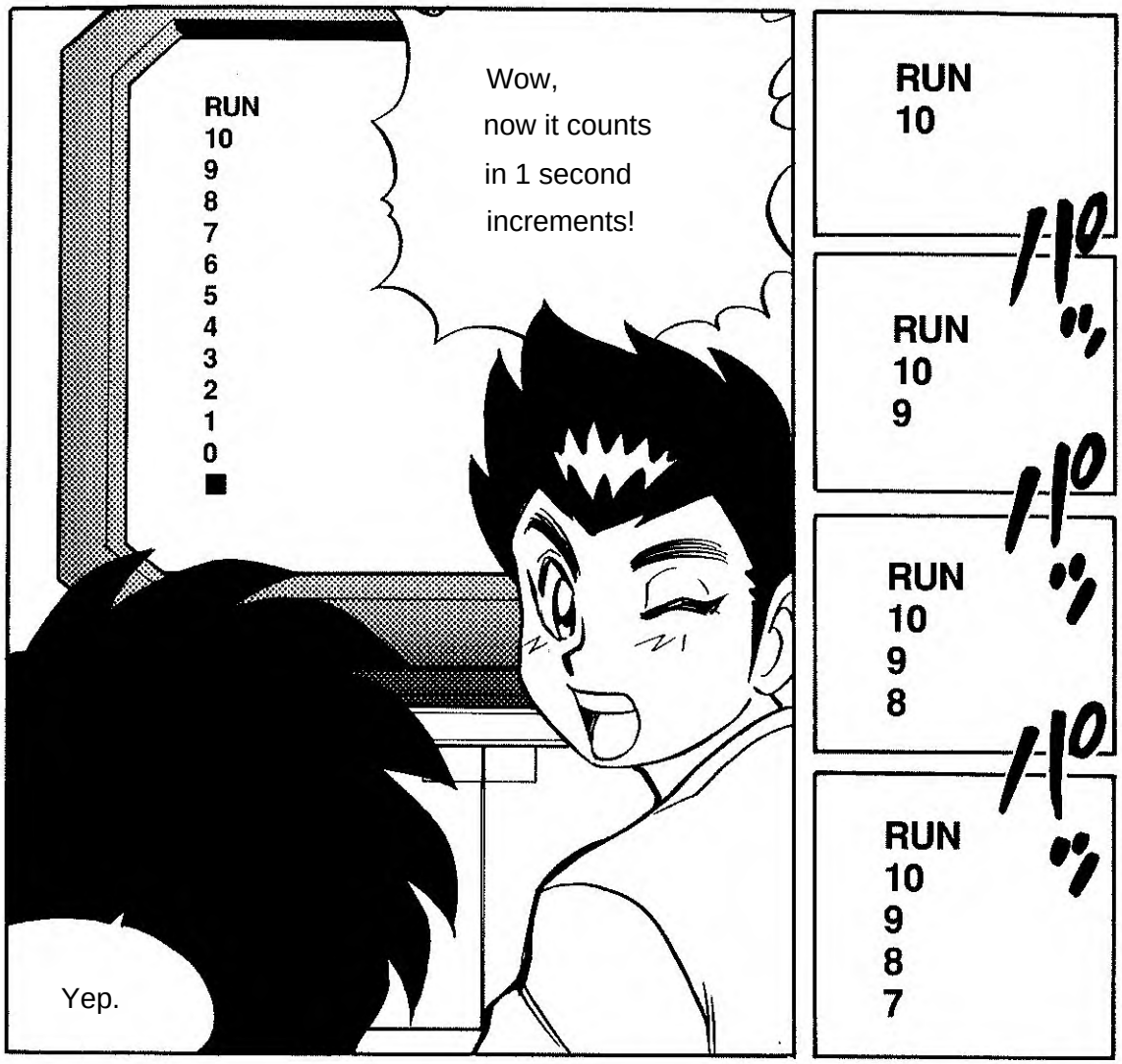
Yeah, it looks like it all happened at once.

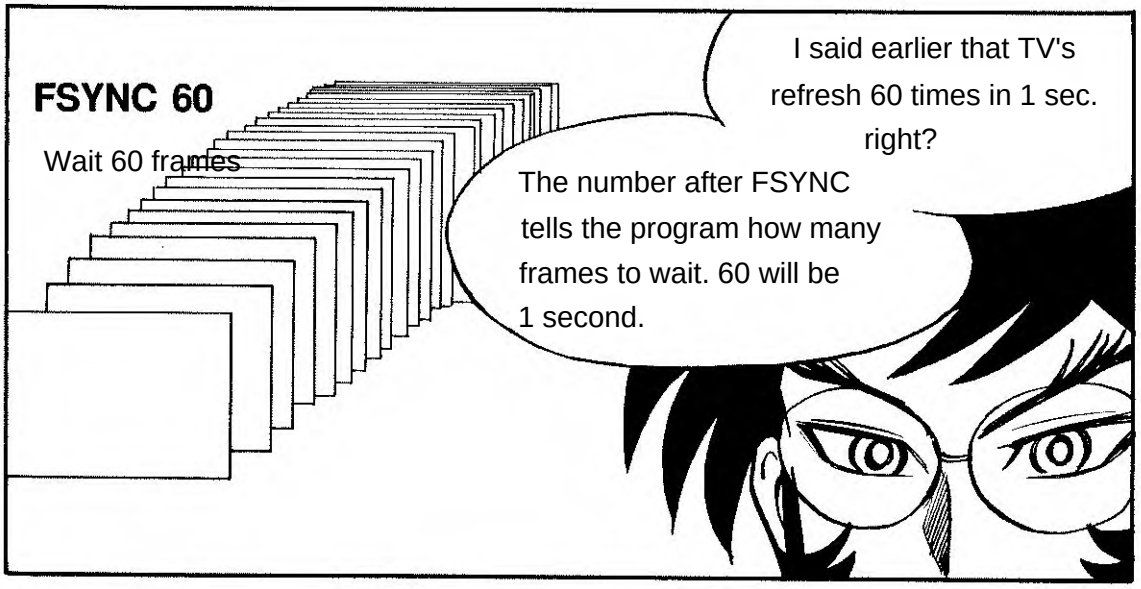


It's a little tough to explain, but Japanese TVs refresh the screen display 60 times per second.

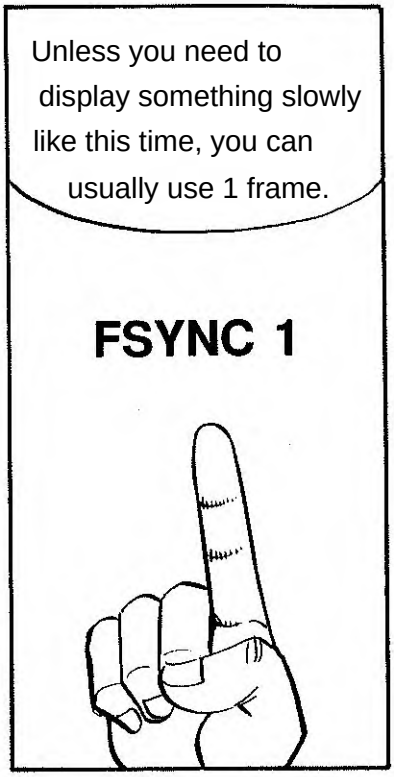
In other words, even if the game data is refreshed at a rate of more than 60 times per second, it will not display properly on a TV.

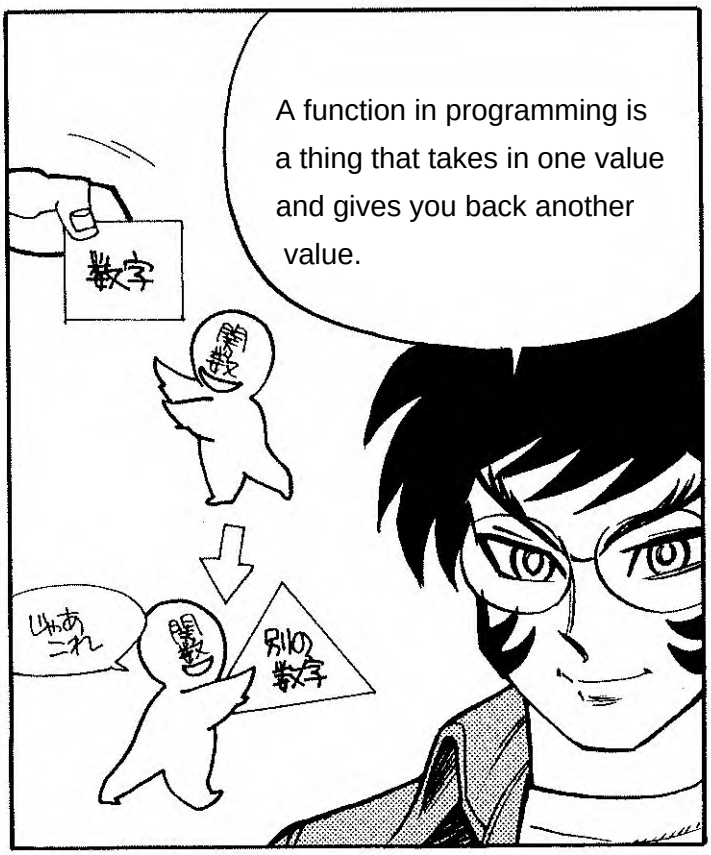

So we need to synchronize the timing with the TV display, using FSYNC.
RUN
In the last program, the entire countdown ends in the time it takes for the TV to refresh the screen once.
RUN
10
9
So that's why it all appeared at once.
I see...
When you execute the FSYNC command, the program pauses and synchronizes with the TV display timing.
Ok.
RUN…
10 FOR A=10 TO 0 S
20 PRINT A
25 FSYNC 60
30 NEXT A
Try running this.
I'll add one line to your count program.
チャ
チャッ

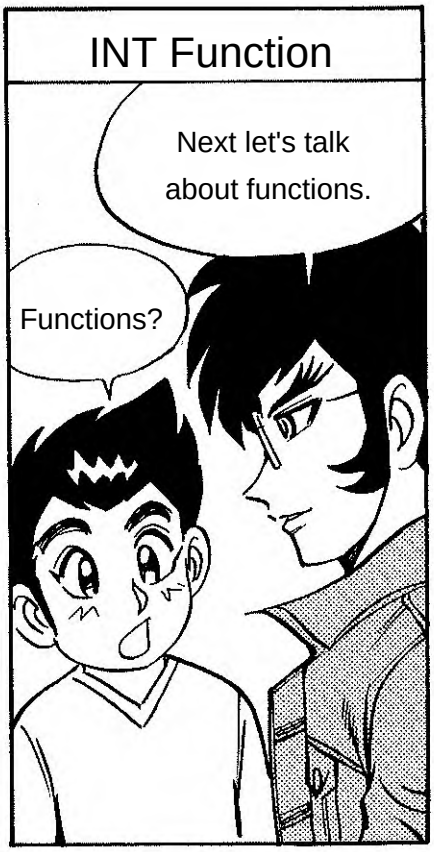

RUN
10
9
8
7
6
5
4
3
2
1
0
Wow,
now it counts
in 1 second
increments!
Yep.
RUN
10
RUN
10
9
RUN
10
9
8
RUN
10
9
8
7

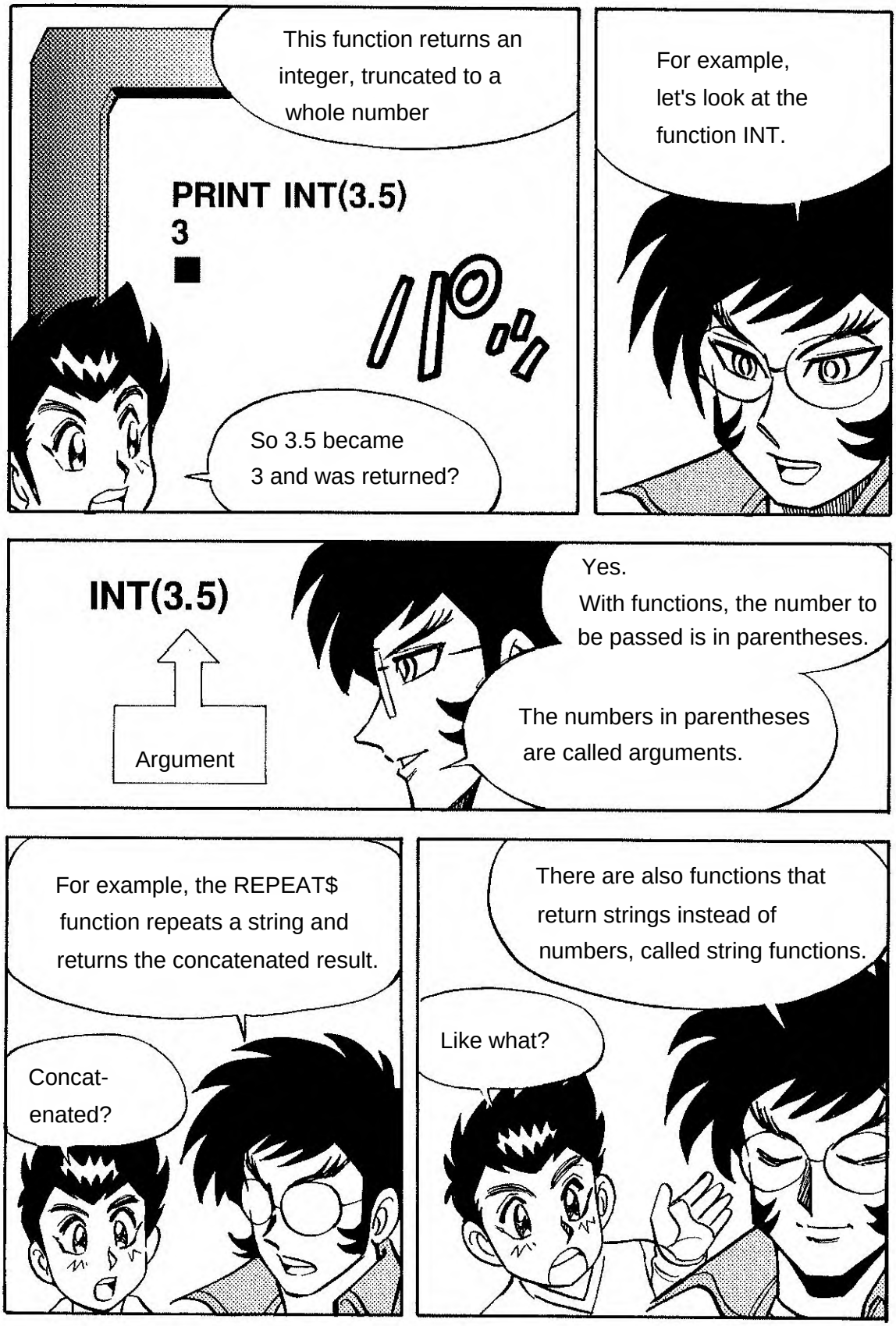

FSYNC 60
Wait 60 frames
I said earlier that TV's refresh 60 times in 1 sec. right?
The number after FSYNC tells the program how many frames to wait. 60 will be 1 second.

In a game's main loop, we typically will use it to match the timing with the display of the TV.
Huh, I see.


Unless you need to display something slowly like this time, you can usually use 1 frame.
FSYNC 1


A function in programming is a thing that takes in one value and gives you back another value.
数字
関数
関数
別の数字


INT Function
Next let's talk about functions.
Functions?


This function returns an integer, truncated to a whole number
PRINT INT(3.5)
3
ﾊﾟｰﾝ
So 3.5 became 3 and was returned?
For example, let's look at the function INT.
INT(3.5)
Argument
Yes.
With functions, the number to be passed is in parentheses.
The numbers in parentheses are called arguments.
For example, the REPEAT$ function repeats a string and returns the concatenated result.
Concat-enated?
There are also functions that return strings instead of numbers, called string functions.
Like what?

It looks like this.

```
PRINT REPEAT$("GO!",5)
GO!GO!GO!GO!GO!
■
```

I see. GO! is connected together five times.

This function takes two arguments. The string to repeat and the number of repeats.

```
A=INT(3.5)
IF A=INT(3.5) THEN ···

A$=REPEAT$("GO!",5)
IF A$=REPEAT$("GO!",5) THEN
```

The numbers and strings returned by a function can be directly assigned to other variables or used in conditional decisions, and can be treated just like variables.

Well, even if you mess up, it will throw an error and you can just rename them then.

OK!

Also, you can't use something used as a name of a function as a variable name.

```
× INT=1
○ INTERNET=1

× REPEAT$="ABC"
○ REP$="ABC"
```

So, REPEAT$ and INT are examples of names that you just can't use yourself.

This function returns a random number from 0 to 1.

It should have equal chances for any value.

```
PRINT RND()
0.448599
OK
■
```

Huh, a totally random number.

Earlier, I said functions need arguments, but some do not.

カタカタ

For example RND does not take anything.

It returns something random so each time it runs it will give something else.

```
PRINT RND()
0.448599
OK
PRINT RND
0.51686
OK
```

Oh, I see. It can work either way.

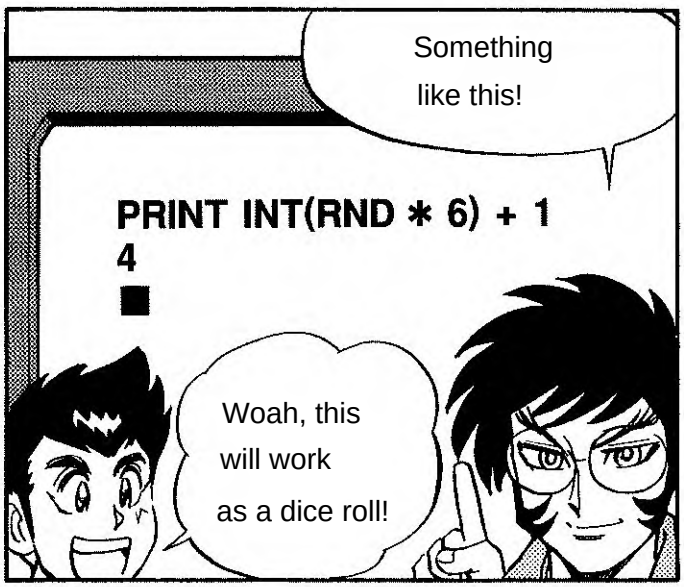
Functions without arguments also don't need parentheses.

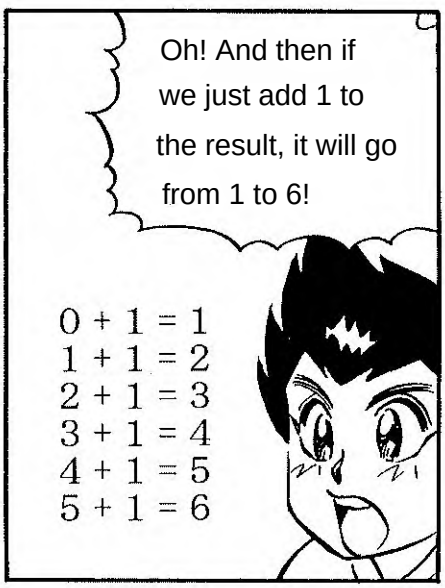
If you multiply the output of RND by 6, the result will be from 0 to 6.

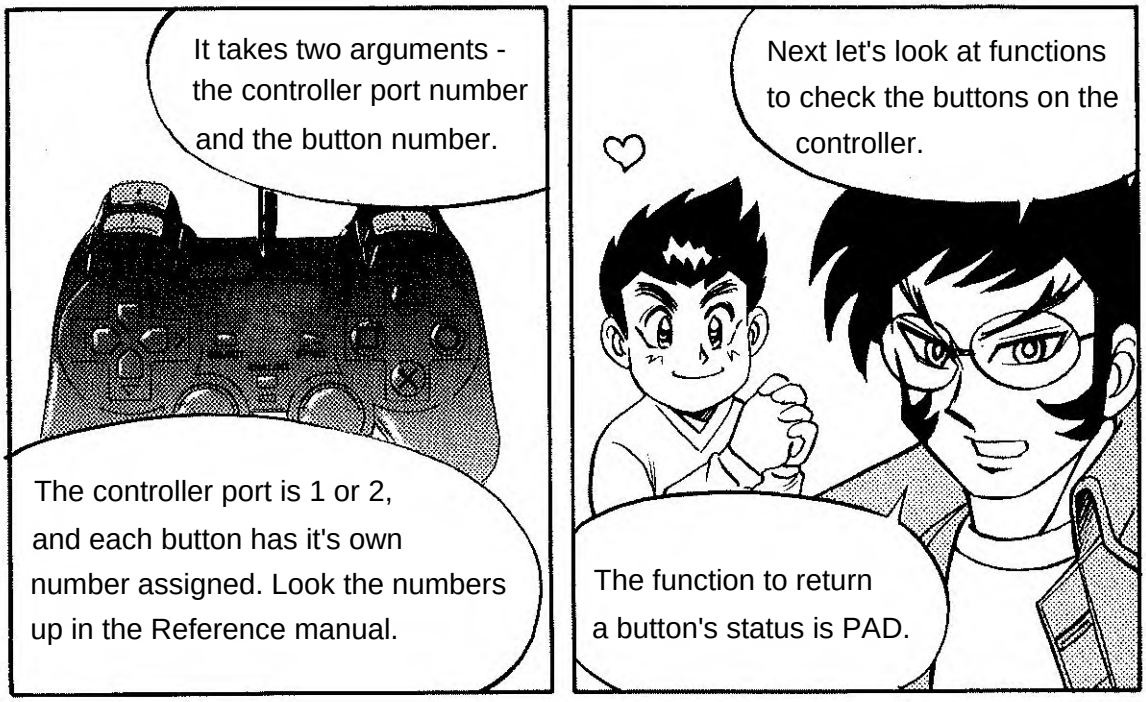
If you then use INT on that as the input, it will discard the decimal and give a number from 0 to 5.

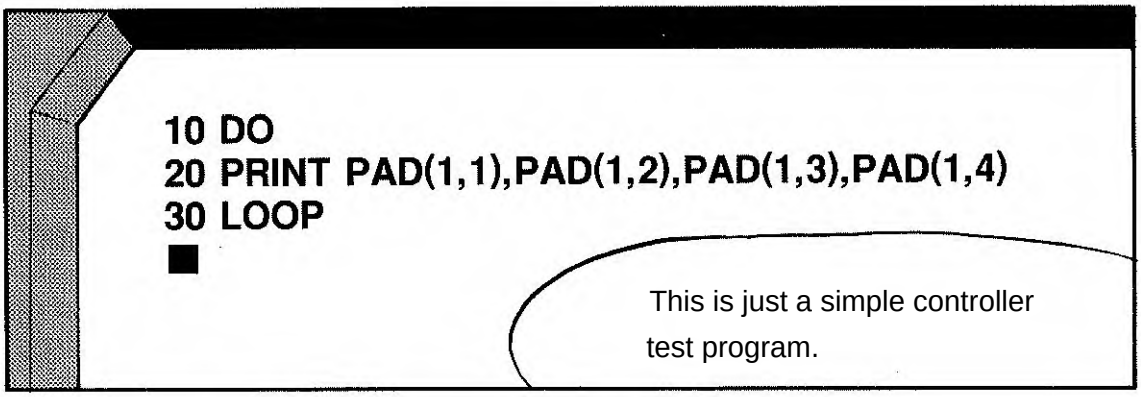
via INT

$$0.0 \times 6 = 0.0 \rightarrow 0$$
$$\vdots$$
$$0.2 \times 6 = 1.2 \rightarrow 1$$
$$\vdots$$
$$\vdots$$
$$0.999 \times 6 = 5.999 \rightarrow 5$$

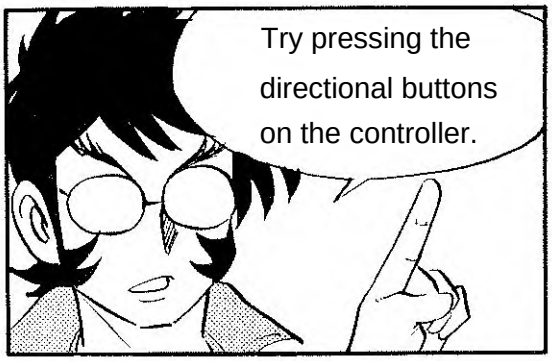


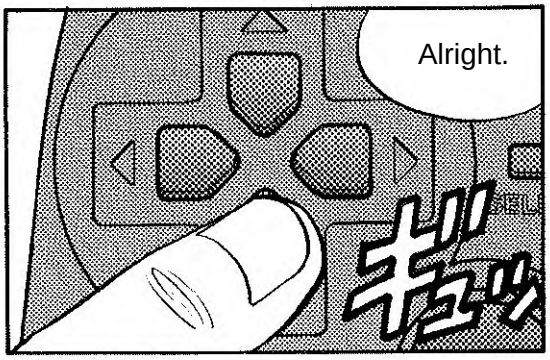


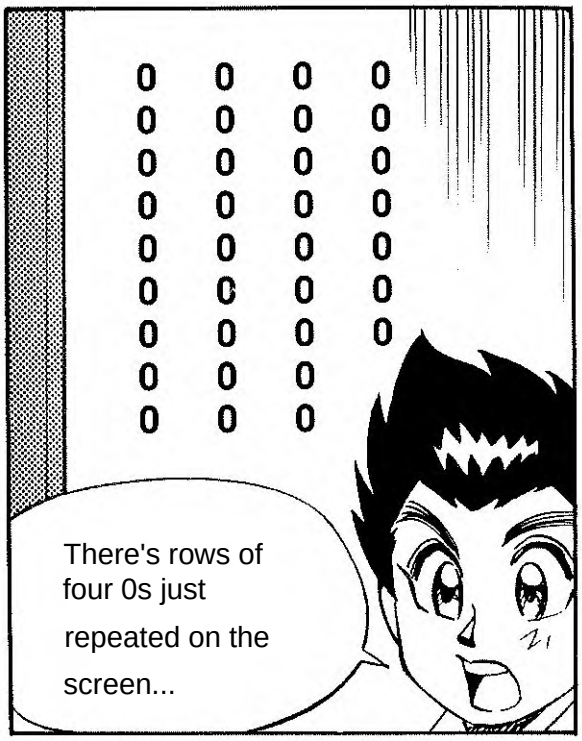

It takes two arguments - the controller port number and the button number.
The controller port is 1 or 2, and each button has it's own number assigned. Look the numbers up in the Reference manual.
Next let's look at functions to check the buttons on the controller.
The function to return a button's status is PAD.

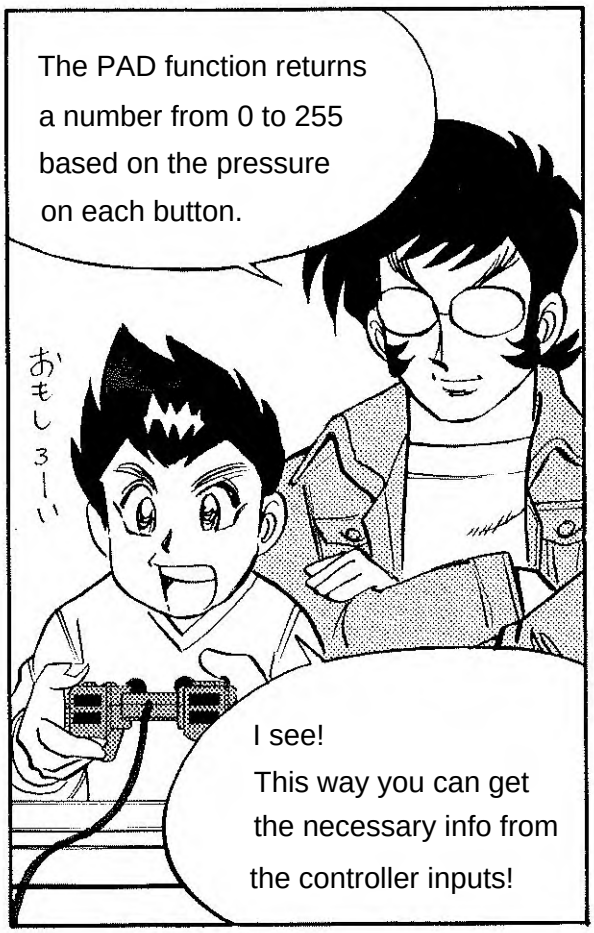

10 DO
20 PRINT PAD(1,1),PAD(1,2),PAD(1,3),PAD(1,4)
30 LOOP
■
This is just a simple controller test program.

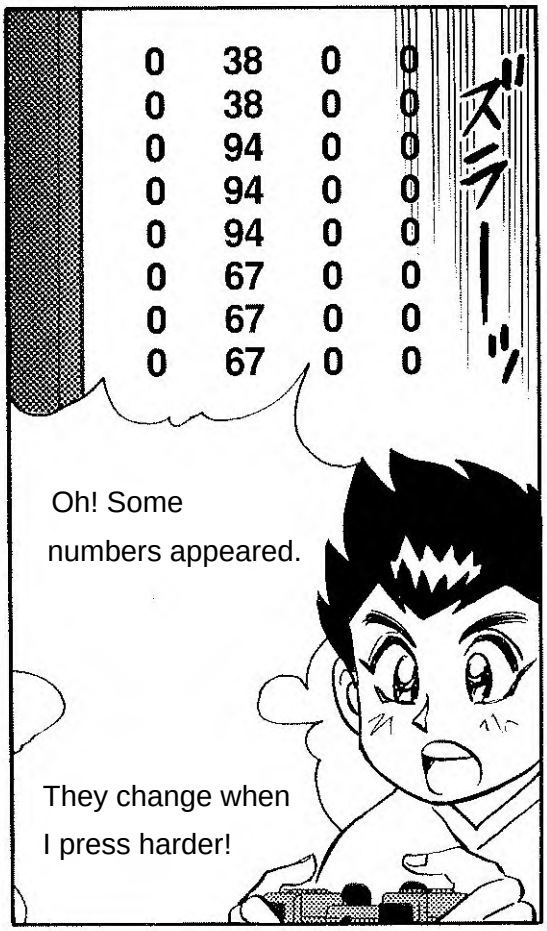
RUN…
Enter...
Try running it.
So this looks at port 1 and checks the status of buttons 1 through 4 and displays the returned values via PAD...


Try pressing the directional buttons on the controller.

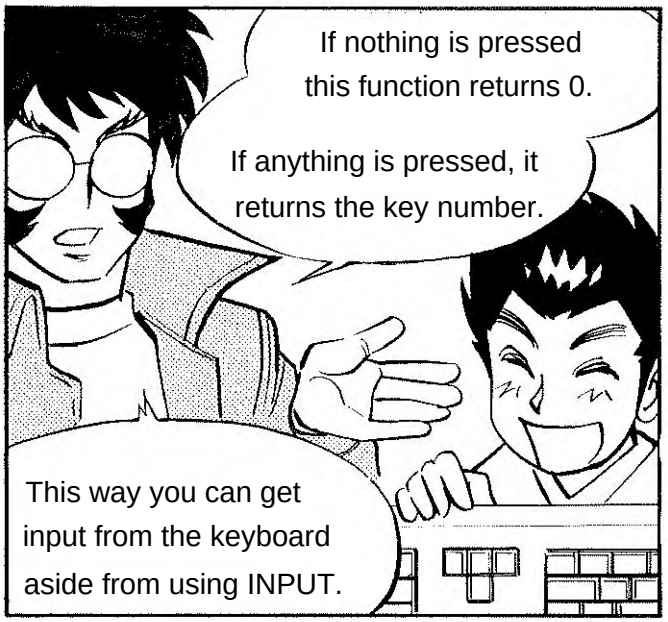

Alright.
キリッ

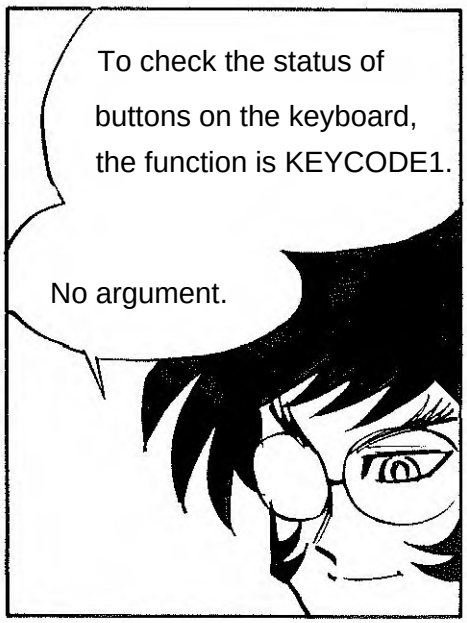

0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0
0 0 0
There's rows of four 0s just repeated on the screen...


The PAD function returns a number from 0 to 255 based on the pressure on each button.
おもしろーい
I see!
This way you can get the necessary info from the controller inputs!

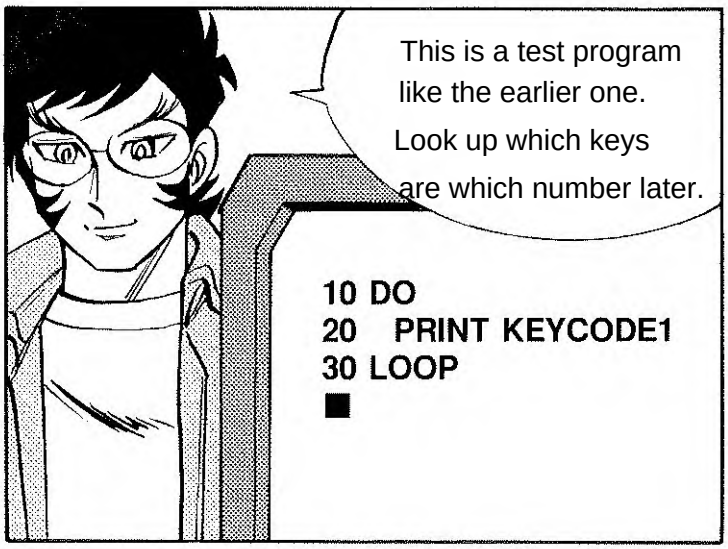

0 38 0 0
0 38 0 0
0 94 0 0
0 94 0 0
0 94 0 0
0 67 0 0
0 67 0 0
0 67 0 0
ズラーッ
Oh! Some numbers appeared.
They change when I press harder!

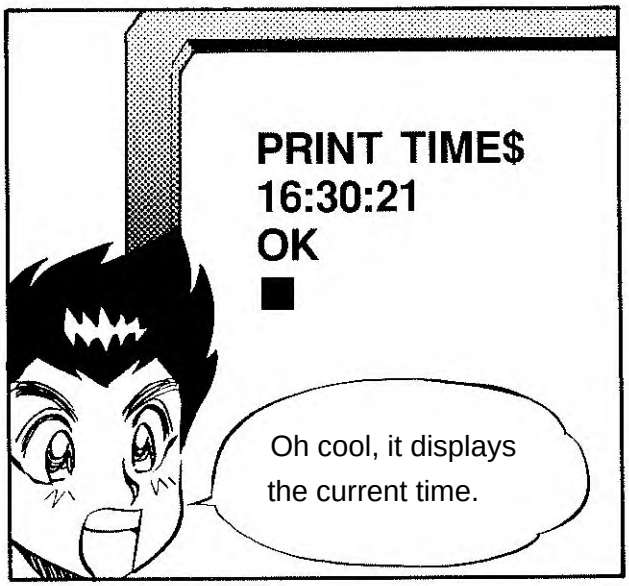
I feel one step closer to making a game!
Whenever you control something with the controller, you use PAD.

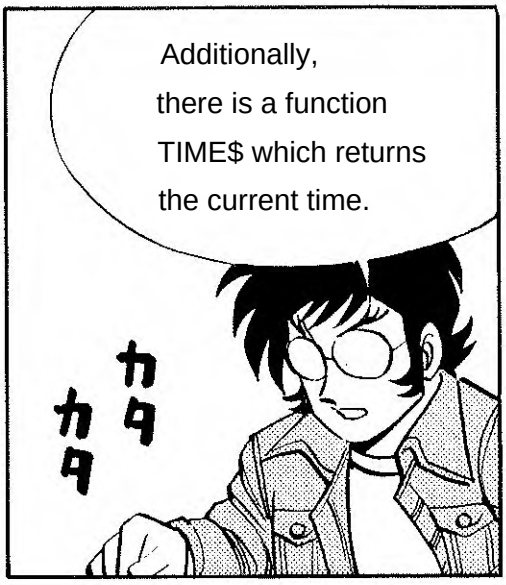

If nothing is pressed this function returns 0.
If anything is pressed, it returns the key number.
This way you can get input from the keyboard aside from using INPUT.


To check the status of buttons on the keyboard, the function is KEYCODE1.
No argument.

I see! Different numbers appear with different keys.
ポチ
ポチッ

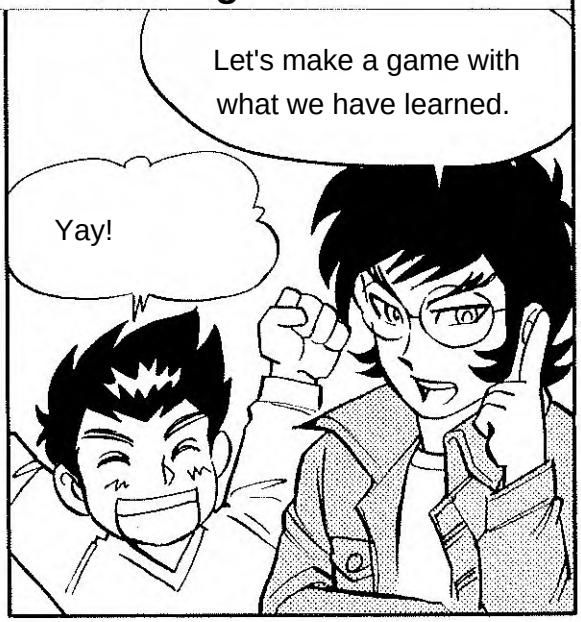

This is a test program like the earlier one. Look up which keys are which number later.
10 DO
20 PRINT KEYCODE1
30 LOOP





Let's Make a Number Guessing Game!

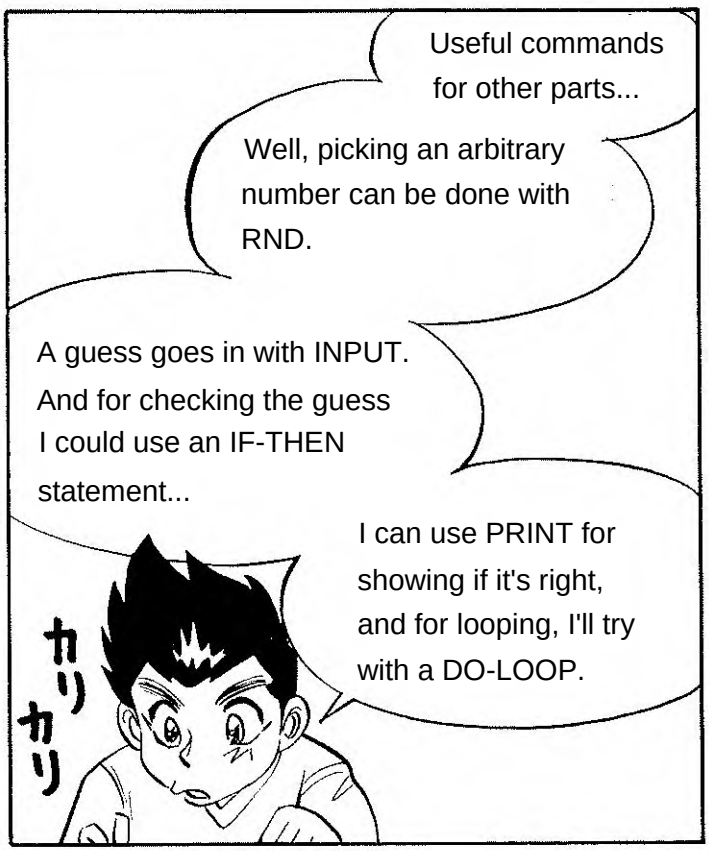
A simple one.
But with the commands we know so far, can you make a game?
We have only learned PRINT to display, so it will be text-based...



Huh... that's it?
It's your first game, so it should be plenty of challenge.
Let's make a guessing game.
A game to guess the number your partner picked.
What!? I've gotta make it on my own?
OK, go for it.
Let's decide the rules.
2
6
15
4
30
97
7
First, a number will be picked.
Then based on the input guess, it should display if it was too big or too small.
?
?
Repeat until the answer is right, then end the game.
Oh, yeah, ok, I'll give it a try.
Of course! What use is it if I make it for you?

* Try yourself before looking at the answer!

1. Pick an arbitrary number
2. The user inputs a guess
3. If it's wrong, display big/small
4. Repeat until it's correct

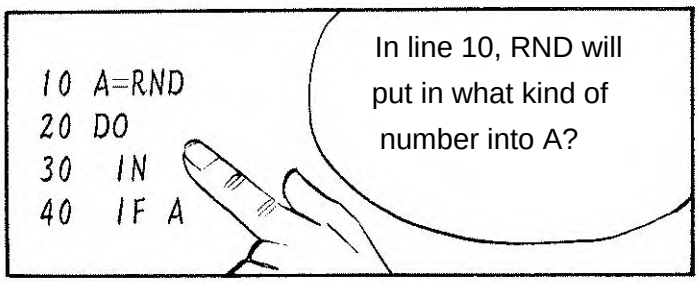

Useful commands for other parts...
Well, picking an arbitrary number can be done with RND.
A guess goes in with INPUT. And for checking the guess I could use an IF-THEN statement...
I can use PRINT for showing if it's right, and for looping, I'll try with a DO-LOOP.
カリ
カリ

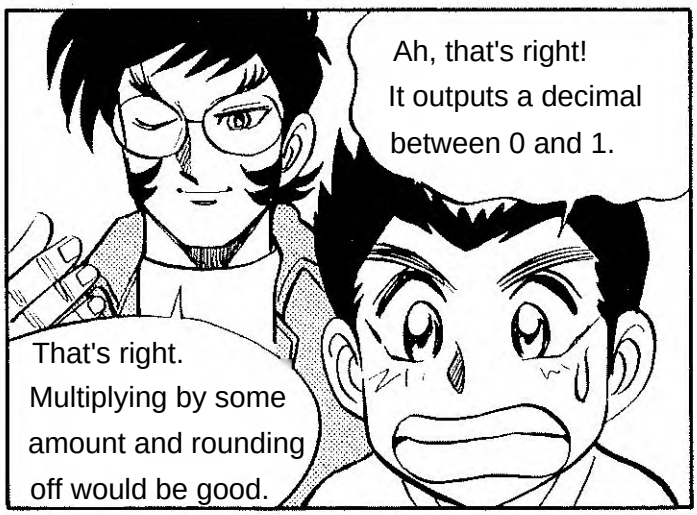
The point that the target number is picked just once while the input is repeated is important.
I see, to repeat, I'll need a DO-LOOP or a FOR-NEXT statement.

```
10 A□ RND
20 DO
30 □ INPUT B
40 □ IF A □ B THEN PRINT “ Bigger"
50 □ IF A □ B THEN PRINT "Smaller"
60 LOOP UNTIL A = B
70 PRINT "CORRECT"
```

□ Check the User's manual for Japanese input

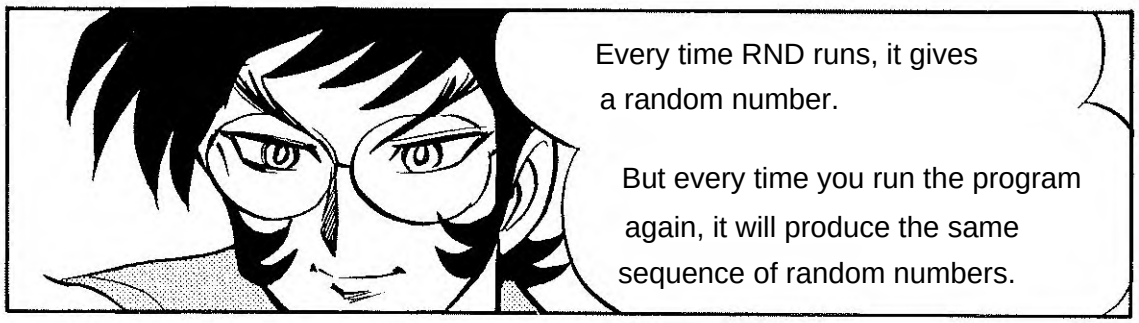
How about this!

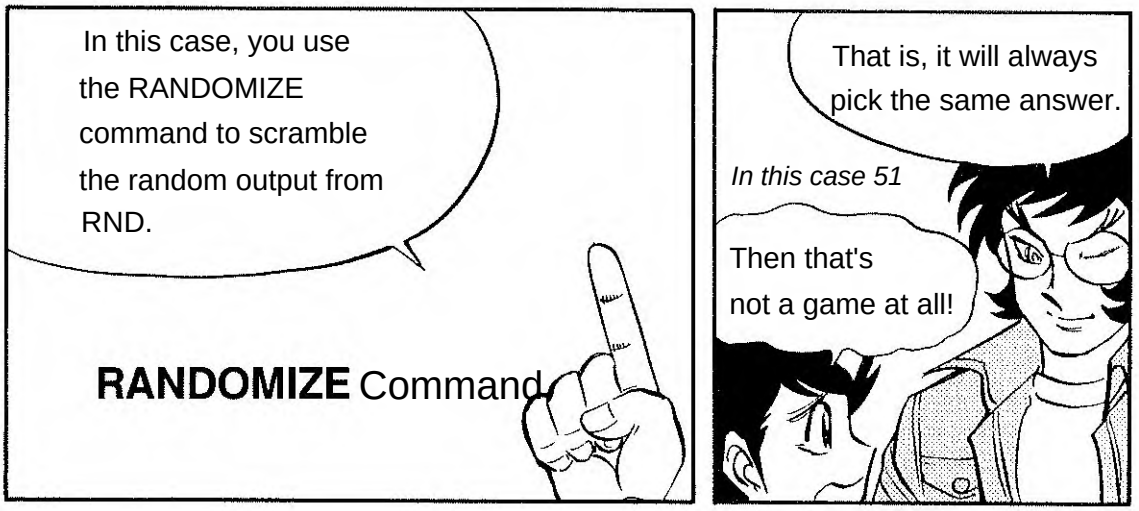




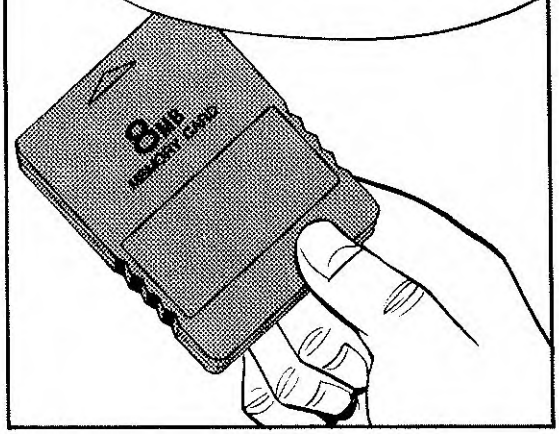
10 A = INT(RND * 100)
20 DO
This should make a number between 0 and 100.
I'll try fixing it up.
タン タン
RUN...
カチャ カチャ
Looks good. Try giving it a run.
50
Bigger
?
Oh, it's bigger than 50.
I'll start with 50.
カチッ
Enter
RUN
?
OK, it's asking for a guess.
?
51
Correct
Got it!
Smaller
?
54
Smaller
?
How about?
カタ カタ
A bit more?
70
Smaller
?
60
Smaller
?
カタ カタ
Dang.
?
70
Smaller
?

Good job.
But there's still a problem.
Huh?
エッヘン
It worked!
Woohoo!


Every time RND runs, it gives a random number.
But every time you run the program again, it will produce the same sequence of random numbers.


In this case, you use the RANDOMIZE command to scramble the random output from RND.
RANDOMIZE Command
That is, it will always pick the same answer.
In this case 51
Then that's not a game at all!

I'll teach you how to save your program onto a "Playstation 2" Memory card (8MB) and then we can call our that our lesson for the day.



If you add it once at the start of your program, it will scramble the output from RND.

```
5 RANDOMIZE
10 A = INT(RND * 100)
20 DO
```

I see. That's a useful command.

If you want to load a program, use the LOAD command

```
LOAD "GAME1"
```

This loads the program "GAME1" from the memory card.

Saving uses the SAVE command.

```
SAVE "GAME1"
```

This saves the current program with the name "GAME1".

The name can have 8 characters, using letters or numbers.

Still a long road ahead. Don't forget to review.

Another step closer to master programmer!

Alright, I've learned how to SAVE and LOAD my work!

Let's write easy-to-read code!

Easy-to-read programs

As the number of lines in a program increases, it can be very difficult to remember which process is on which line. You may also forget how to use variables or what a subprogram does. Here, we will introduce how to write programs that are easy to read and understand, in order to reduce these problems as much as possible.

Let's improve the writing style

* Indentation

• Inserting a space at the beginning of the line is called indentation. Indentation can be used to help make repeated parts or subprograms easier to read.

Ex:

Without indenting

```
100 DO
110 CLEAR
120 FOR A=1 TO 10
130 IF PADQ,A>0 THEN
140 PRINT "BUTTON";A,"ON"
150 END IF
160 NEXT A
170 FSYNC 1
180 LOOP UNTIL PAD(1,20)=1
```

With indenting

```
100 DO
110  CLEAR
120  FOR A=1 TO 10
130   IF PAD(1,A)>0 THEN
140    PRINT "BUTTON";A,"ON"
150   END IF
160  NEXT A
170  FSYNC 1
180 LOOP UNTIL PAD(1,20)=1
```

Both of these programs have exactly the same content, but without indentation it would be difficult to see at a glance which lines are being repeated by the FOR~NEXT command or which series of lines are being repeated by the DO~LOOP.
On the other hand, indenting makes it easier to see at a glance which lines are repeated with FOR~NEXT commands.
For example, you can see that the part repeated with the FOR~NEXT commands in lines 120 to 160 is indented by one or more characters from the FOR line.

• Indentation can be used for commands that deal with repetition or grouping of programs, such as DO~LOOP commands, IF THEN~END IF commands, SELECT CASE-END SELECT commands, and subprograms

Comments

• Comments include the REM command and the end-of-line comment (! command).
These annotations are ignored when a program is executed, but by utilizing annotations you can write programs that are easier to read

♦Use comments as a note-keeping tool

•Using the REM command to add comments about the program's functions and operations makes it easier to understand what the program is about.

•Using end-of-line comments (the ! command) to comment out what a line does makes it easier to understand.

Ex:

No comments

```
100 D0
110 IF PAD(1,6)>0 THEN EXIT DO
120 LOOP
```

With comments

```
100 REM Wait for the O button to be pressed
 110 DO
120  IF PAD(1.6)>0 THEN EXIT DO !If O is pressed then loop
130 LOOP
```

♦Commenting out

•You can also prevent a line that performs a certain process from being executed by making it a comment using an end-of-line comment (the ! command). This can be used to prevent a program from being executed without deleting it.

Ex ▫

```
100 DO
110  CLEAR
120 !IF PAD(1,7)>0 THEN
130  PRINT "HAPPY"
140 !END IF
150  FSYNC 1
160 LOOP
```

In this example, lines 120 and 140 are commented out to prevent them from being executed. This means that line 130 will always be executed (without executing the IF THEN command) even if you do not press the X button.
If you want it to display while you hold down the x button, you can simply disable the endnotes, saving you the trouble of re-entering your program.

Let's improve the writing style

♦Easy to understand variable names

•Programs will be easier to understand if you give variables names that allow you to immediately imagine how they will be used. For example, it is easier to imagine uses for SCORE=SCORE+300 than for A=A+300.

•It can also be applied to the number of elements in an array, etc.

Ex:

Making a 2*8 array

```
100 DIM WPN(Z,8)   The use is unclear
```

Assign values o variables before declaring the array

```
100 PLAYER=2
110 WEAP0NS=8
120 DIM WPN(PLAYER,WEAPONS) You can imagine the use of each dimension.
```

♦Making use of Sub-programs

•If a program contains multiple similar processes, grouping them together as subprograms will reduce the amount of program code and make it easier to read overall.
Also, if built as a subprogram, you only need to modify the subprogram, saving you the trouble of modifying multiple lines.

• Grouping certain functions into subprograms makes them easier to handle. For example, classifying functions into subprograms such as initialization and updating character coordinates makes the program easier to understand.

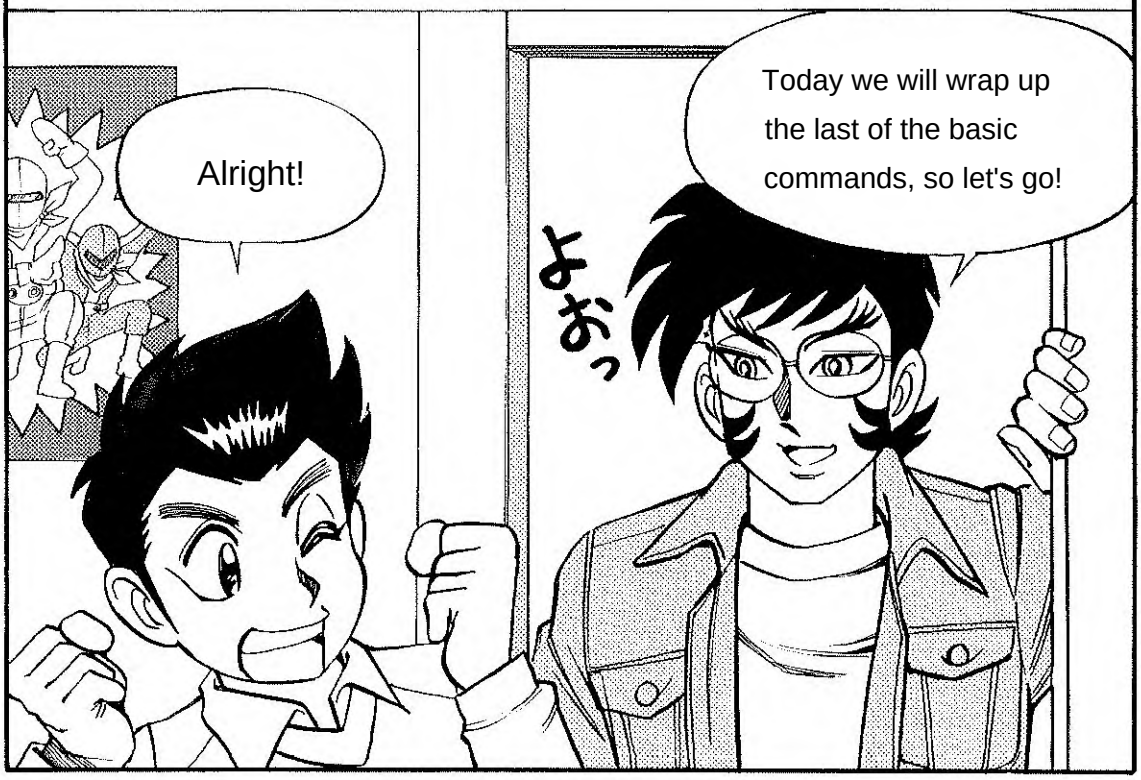
Ex:

```
100 D0 ! Main Loop
110  CALL PAD_CHK(1)!Controller input check
120  CALL UPDATE_SHIP !Update your ship
130  FOR A=1 TO 100
140   CALL UPDATE_ZAKO(A) !Update other characters
150  NEXT A
160 LOOP
```

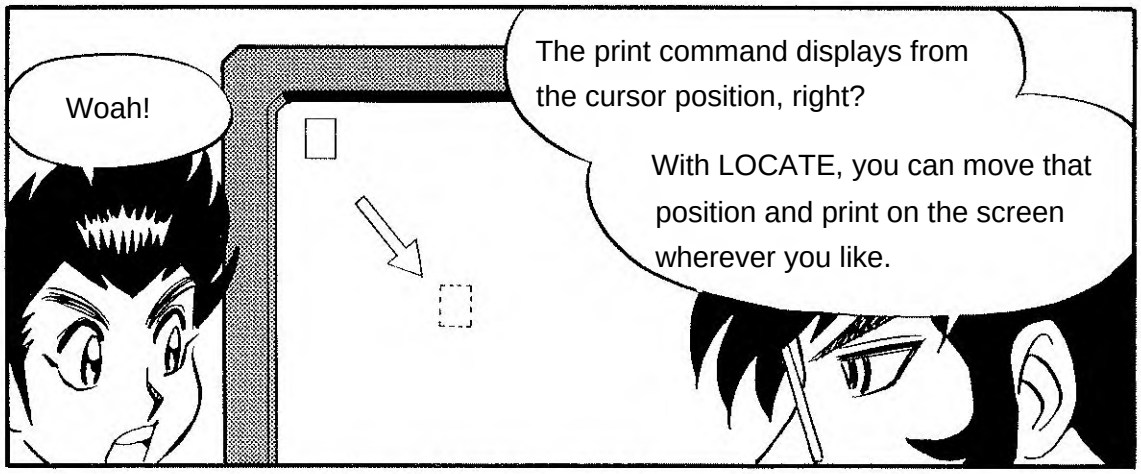
In the example, each subprogram defined for each function is called in order to make it easier to understand the main loop (the repetitive operations that are the core of the program).

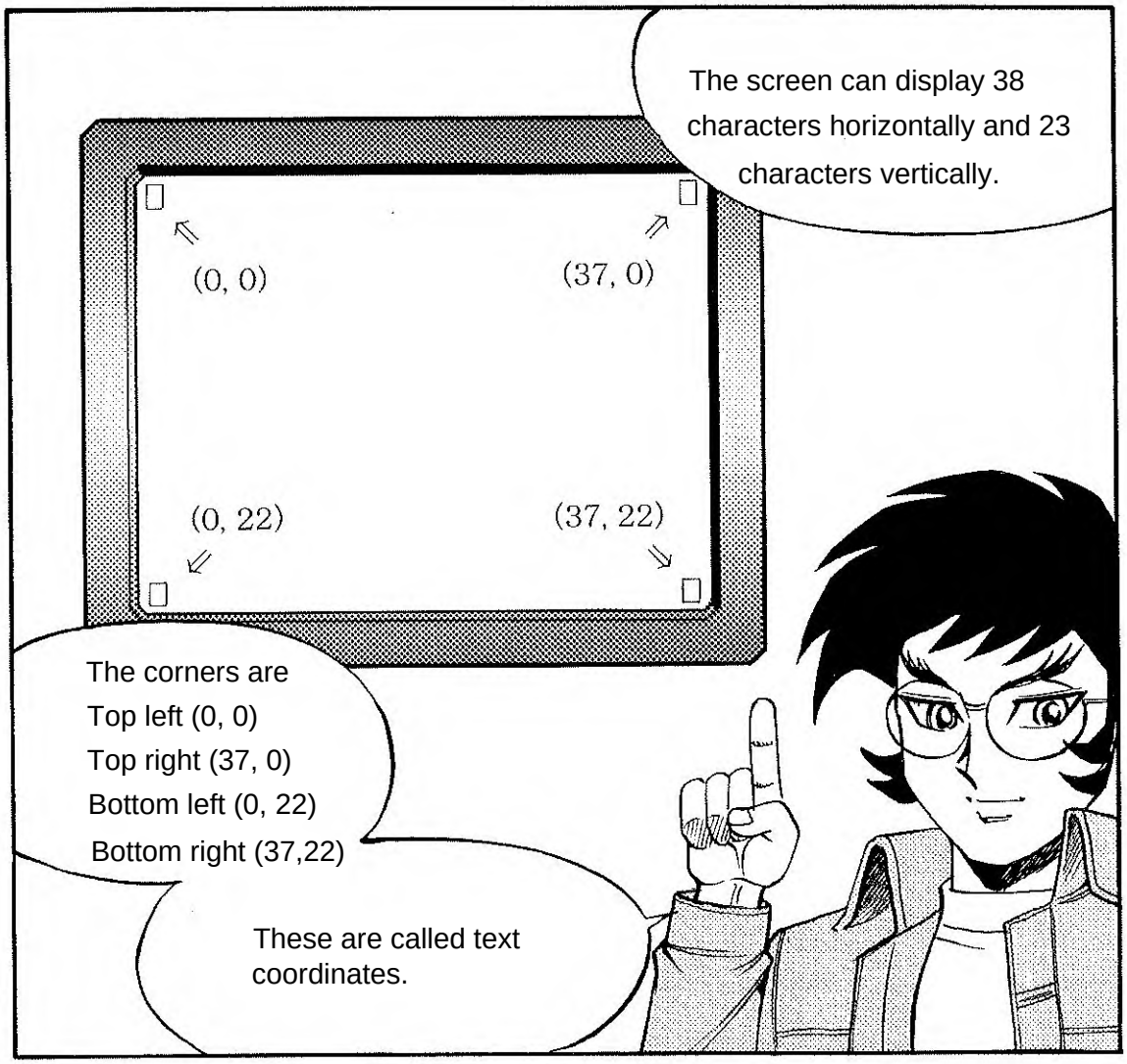
♦Subprogram names will also be easier to understand if you give them self-apparent names.
In addition, please refer to the sample programs and demo programs to create a program that is easy to read.

Basic Commands

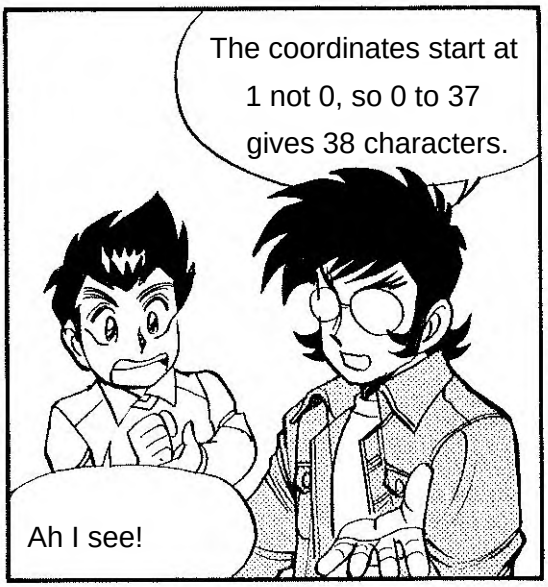


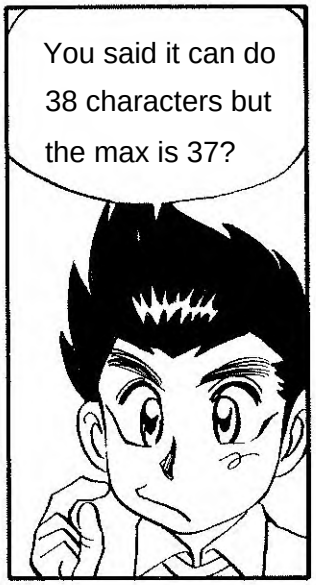
First I will teach you the LOCATE command.
You use this before a PRINT command to freely control the position of the cursor.

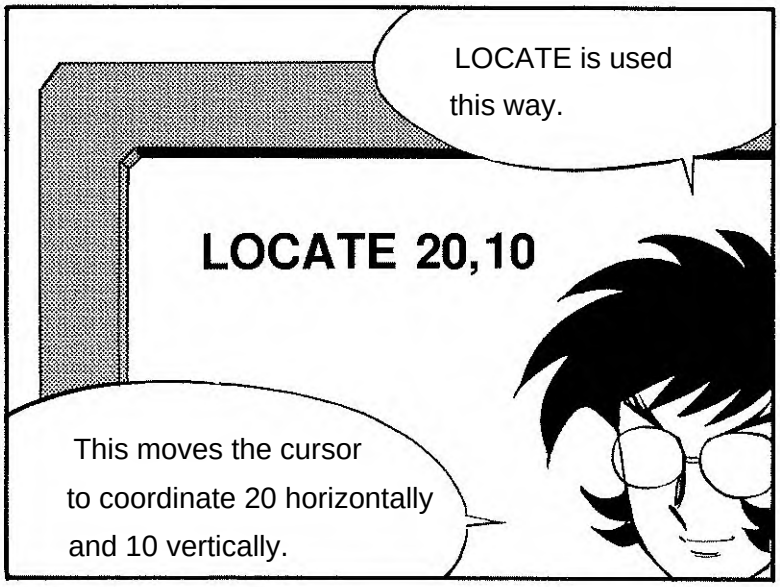

Woah!
The print command displays from the cursor position, right?
With LOCATE, you can move that position and print on the screen wherever you like.

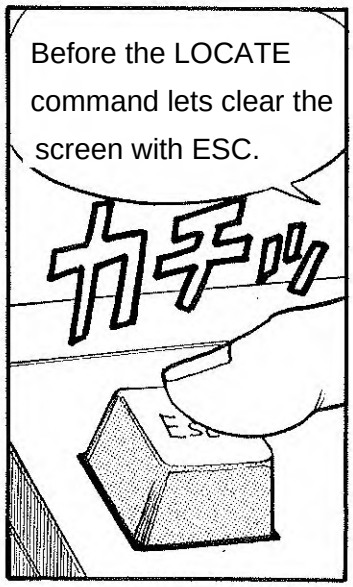

The screen can display 38 characters horizontally and 23 characters vertically.
(0, 0)
(37, 0)
(0, 22)
(37, 22)
The corners are
Top left (0, 0)
Top right (37, 0)
Bottom left (0, 22)
Bottom right (37,22)
These are called text coordinates.

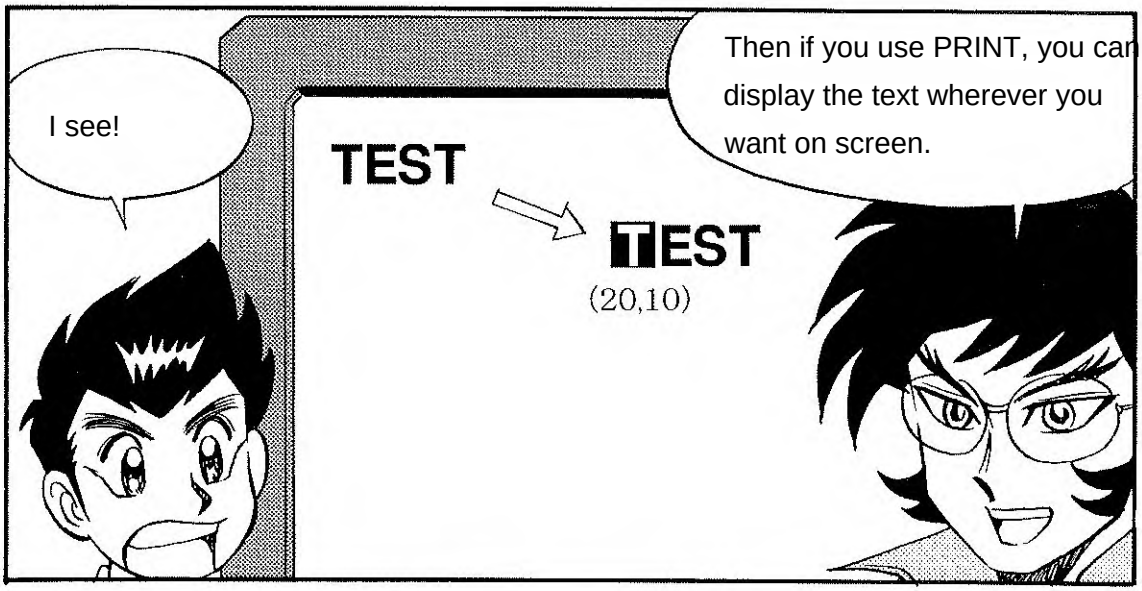
There is a command to raise the width, but we will get to that later.

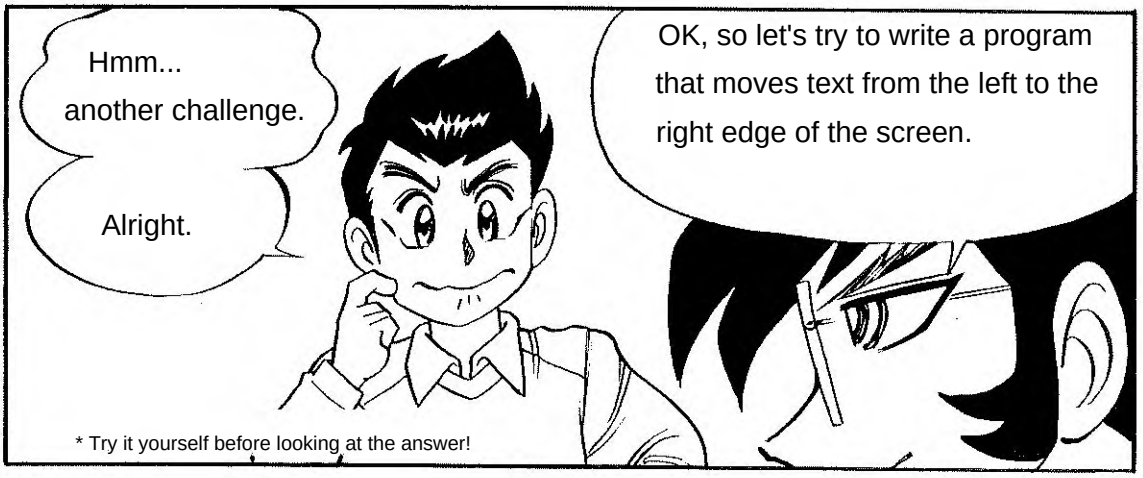

The coordinates start at 1 not 0, so 0 to 37 gives 38 characters.
Ah I see!


You said it can do 38 characters but the max is 37?


LOCATE is used this way.
LOCATE 20,10
This moves the cursor to coordinate 20 horizontally and 10 vertically.


Before the LOCATE command lets clear the screen with ESC.
カチッ
ESC


I see!
TEST
TEST
(20,10)
Then if you use PRINT, you can display the text wherever you want on screen.


Hmm... another challenge.
Alright.
OK, so let's try to write a program that moves text from the left to the right edge of the screen.
* Try it yourself before looking at the answer!

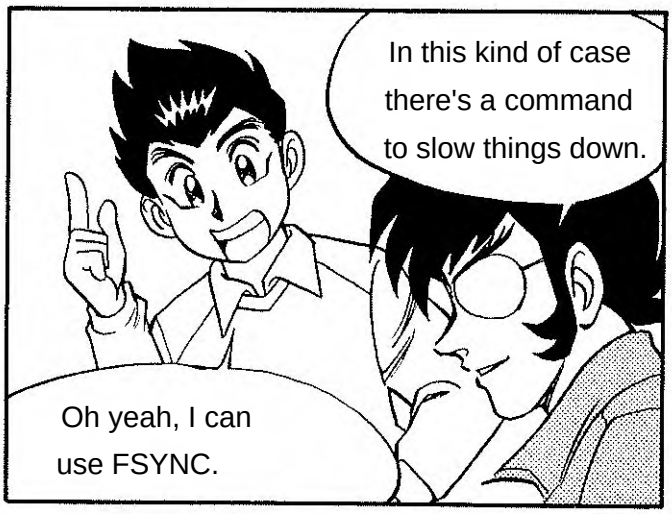
I'll just try something and print out TEST first.
10 A=0
20 LOCATE A,0
30 PRINT "TEST"
So then...
Well, we use LOCATE to set the cursor position... and we're going to move it, so I'll use variables...
Make the location a variable and use LOCATE to move the cursor to the location of the variable.
By repeatedly adding 1 I'll change the position.
Make a DO loop to repeat...
カチャ
カチャ
Give it a go.
RUN...
カチャ
Add in DO on line 15 Keep adding to the variable while looping.
10 A=0
15 DO
20 LOCATE A,0
30 PRINT "TEST"
40 A=A+1
50 LOOP
How's this?

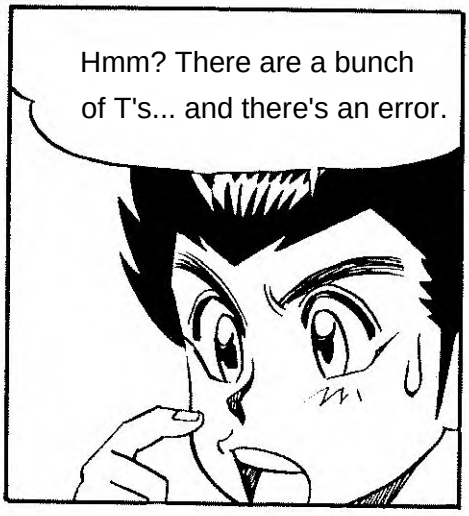
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
EST ...now running.

20 LOCATE A,0

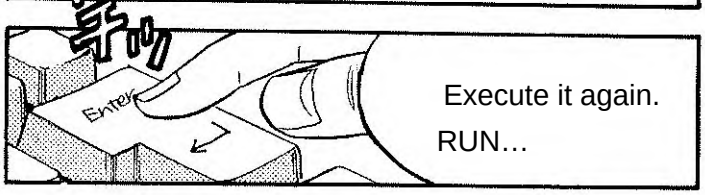
Error in specified coordinates.

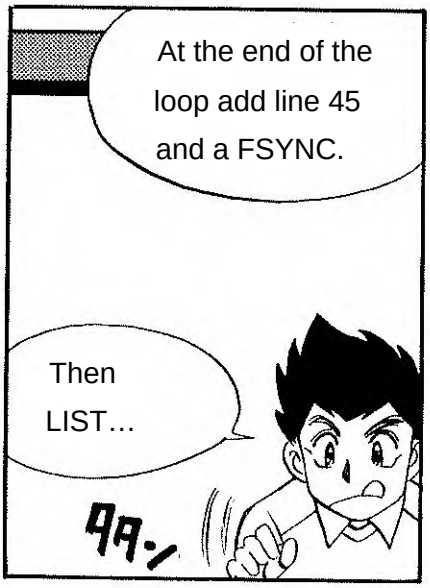
The coordinates specified are outside the screen area.



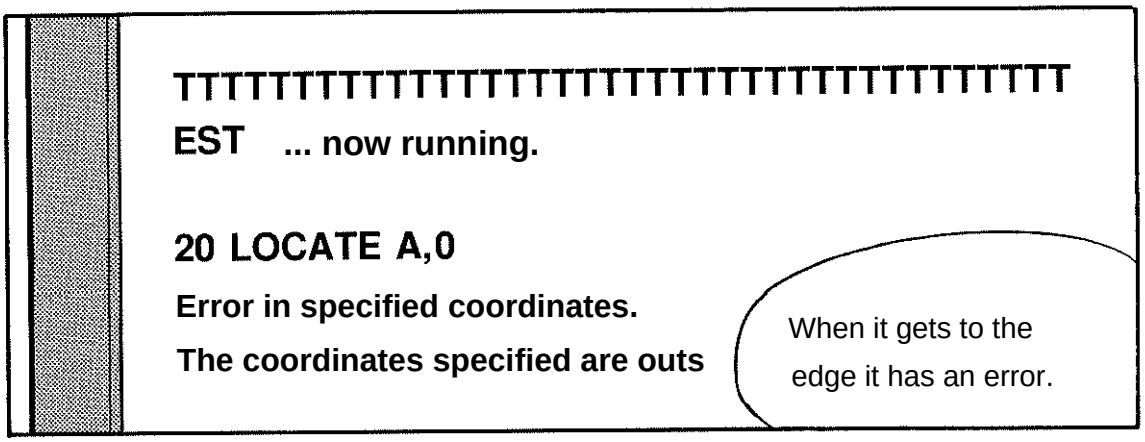


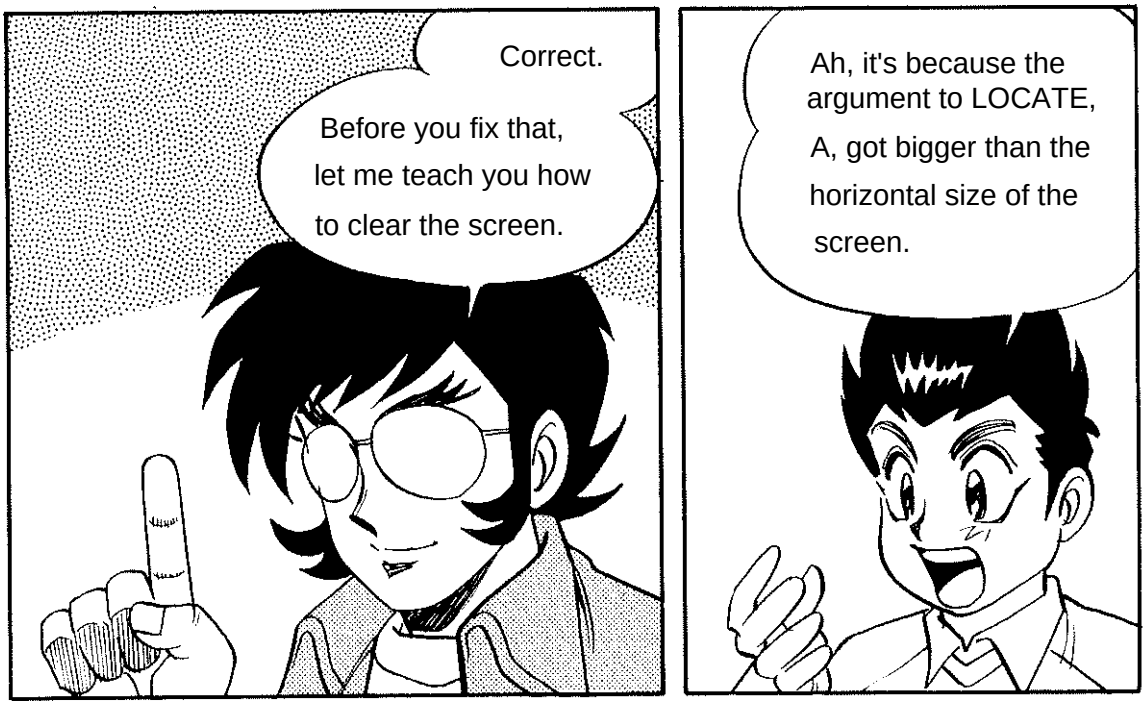

```
10 A=0
15 DO
20   LOCATE A,0
30   PRINT "TEST"
40   A=A+1
45   FSYNC 60
50 LOOP
```

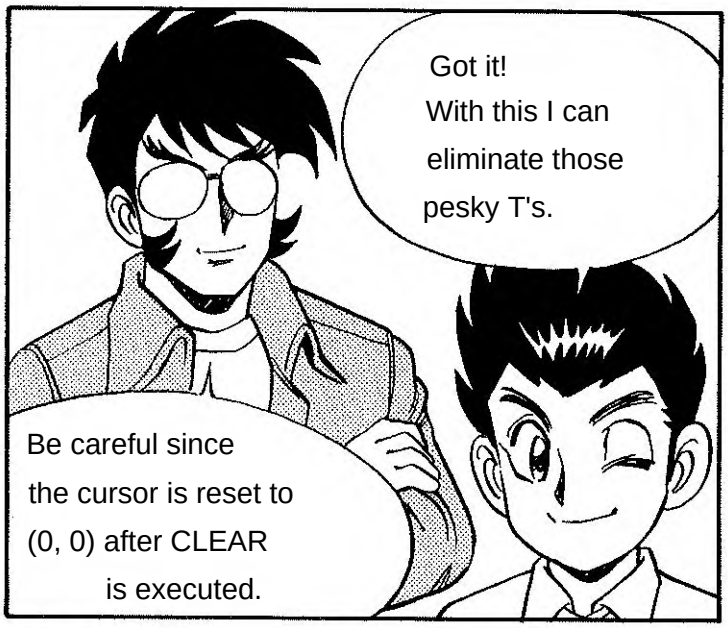


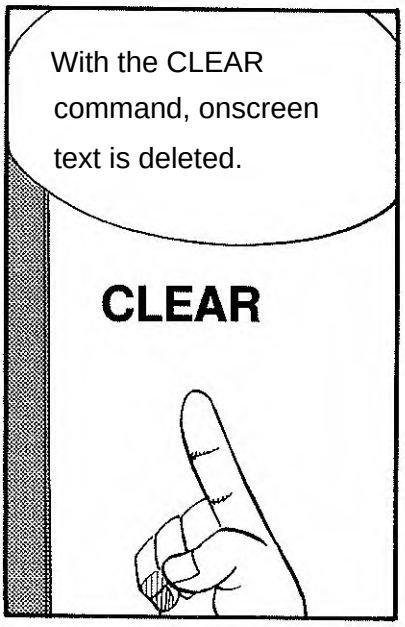


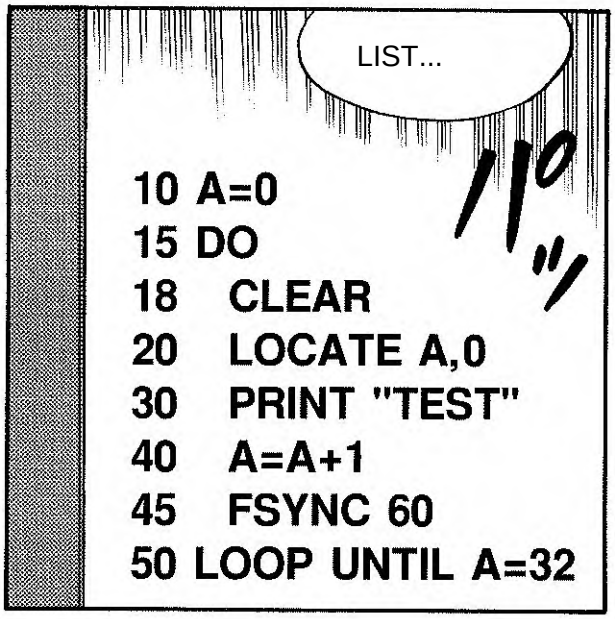
Oh I see... The line of T's are leftover from each copy of TEST that prints.
TTEST
TTTTTTTEST
OK it's running slowly now.

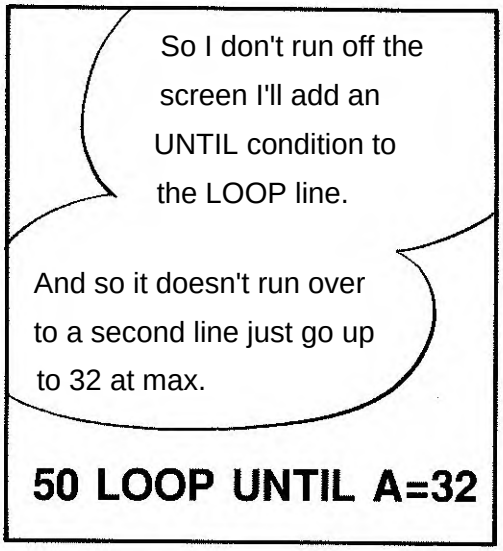

TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
EST ... now running.
20 LOCATE A,0
Error in specified coordinates.
The coordinates specified are outs
When it gets to the edge it has an error.


Correct.
Before you fix that, let me teach you how to clear the screen.
Ah, it's because the argument to LOCATE, A, got bigger than the horizontal size of the screen.

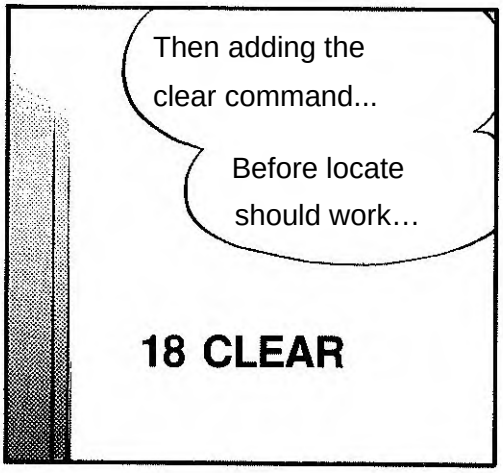

Got it!
With this I can eliminate those pesky T's.
Be careful since the cursor is reset to (0, 0) after CLEAR is executed.

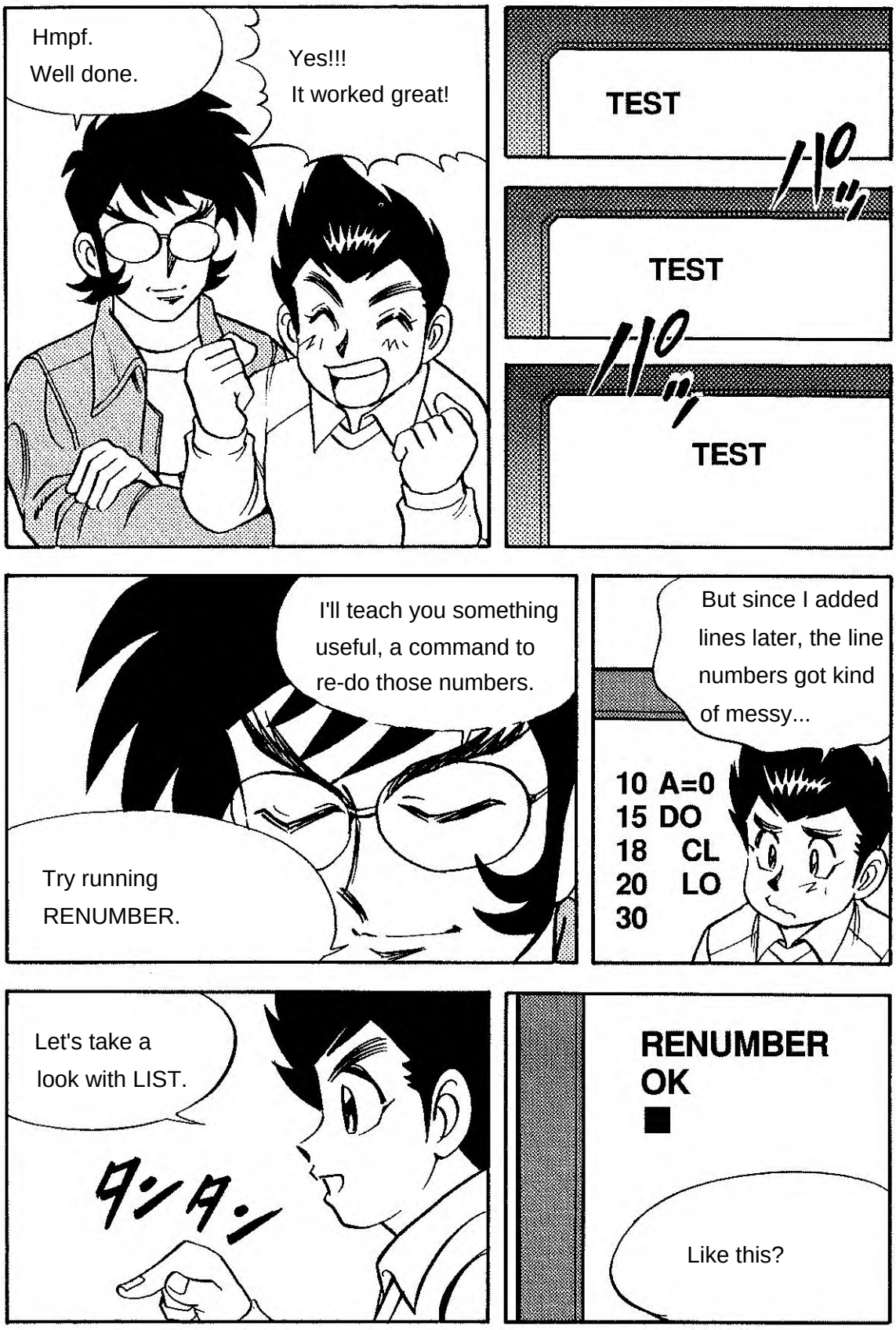

With the CLEAR command, onscreen text is deleted.
CLEAR

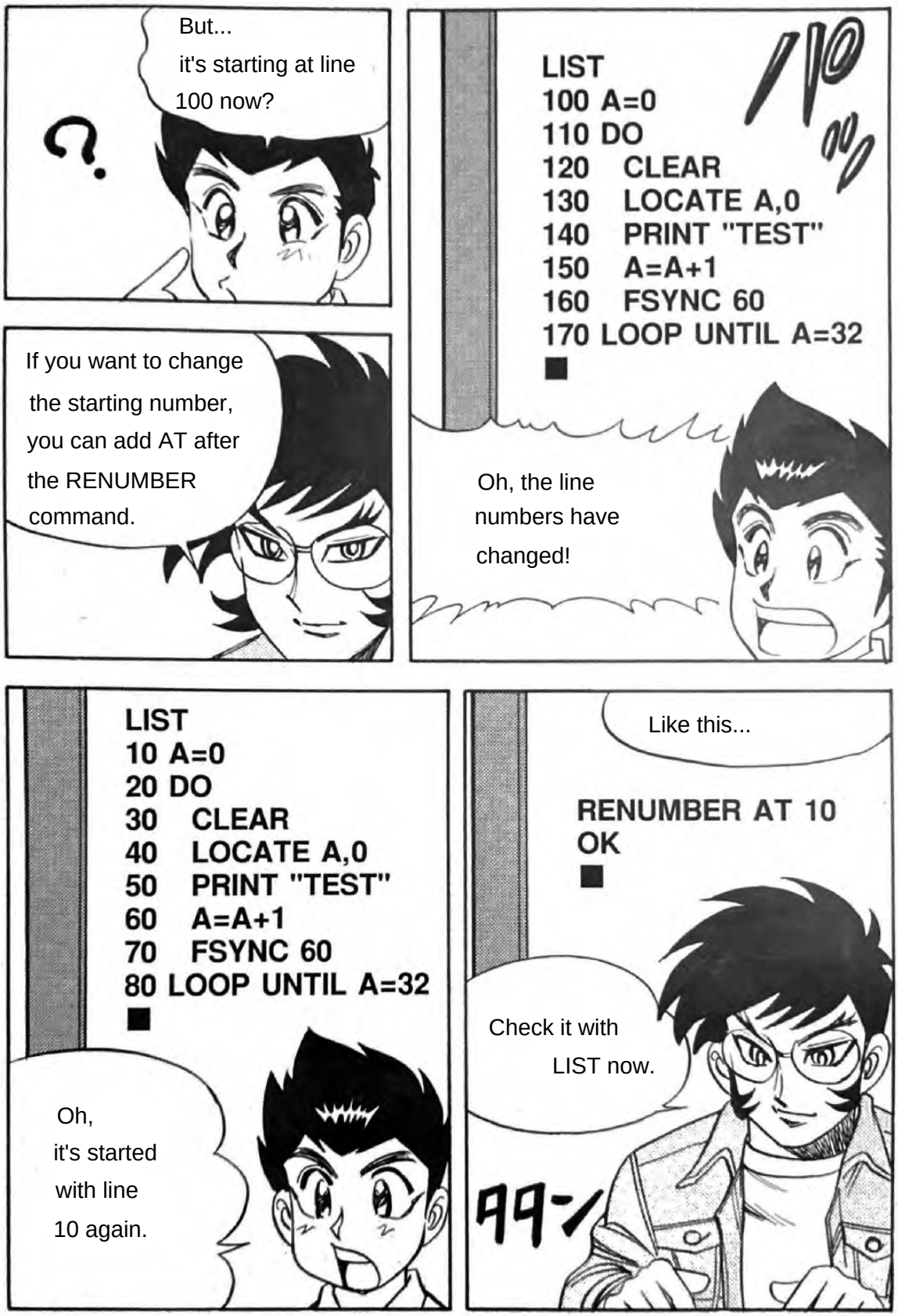

LIST...
10 A=0
15 DO
18 CLEAR
20 LOCATE A,0
30 PRINT "TEST"
40 A=A+1
45 FSYNC 60
50 LOOP UNTIL A=32


So I don't run off the screen I'll add an UNTIL condition to the LOOP line.
And so it doesn't run over to a second line just go up to 32 at max.
50 LOOP UNTIL A=32

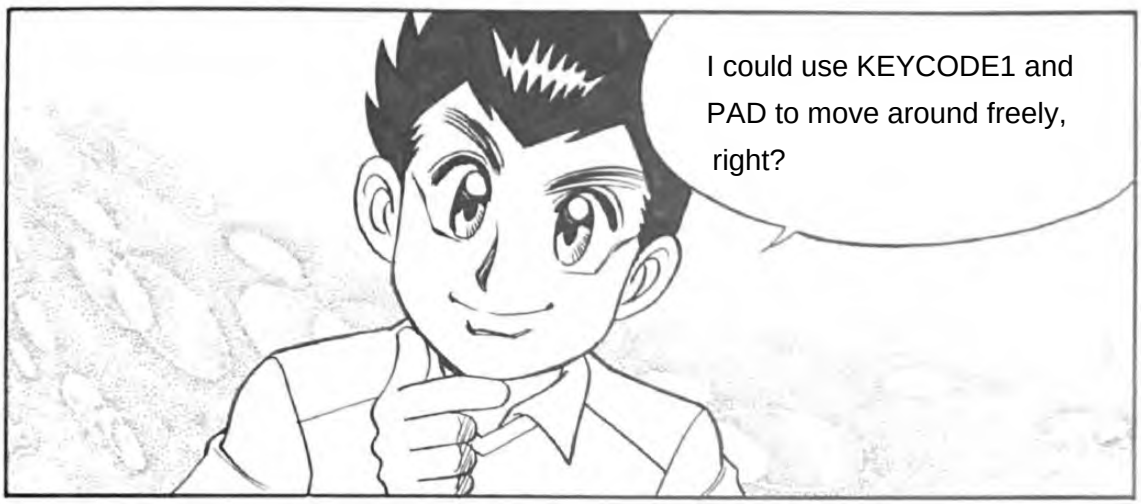
Maybe now it works...
RUN…
カチャカチャ

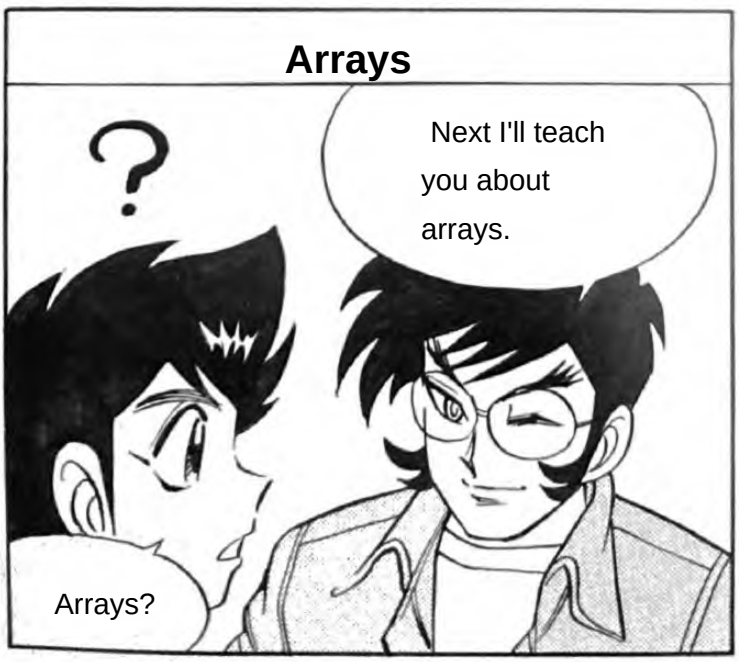

Then adding the clear command...
Before locate should work…
18 CLEAR


Hmpf.
Well done.
Yes!!!
It worked great!
TEST
TEST
TEST
I'll teach you something useful, a command to re-do those numbers.
Try running RENUMBER.
But since I added lines later, the line numbers got kind of messy...
10 A=0
15 DO
18 CL
20 LO
30
Let's take a look with LIST.
RENUMBER
OK
Like this?


But...
it's starting at line 100 now?
If you want to change the starting number, you can add AT after the RENUMBER command.
LIST
100 A=0
110 DO
120 CLEAR
130 LOCATE A,0
140 PRINT "TEST"
150 A=A+1
160 FSYNC 60
170 LOOP UNTIL A=32
■
Oh, the line numbers have changed!
LIST
10 A=0
20 DO
30 CLEAR
40 LOCATE A,0
50 PRINT "TEST"
60 A=A+1
70 FSYNC 60
80 LOOP UNTIL A=32
■
Oh, it's started with line 10 again.
Like this...
RENUMBER AT 10
OK
■
Check it with LIST now.

In text games, you often will use a program like what we wrote to handle enemy and player movement.


I could use KEYCODE1 and PAD to move around freely, right?


Arrays
?
Next I'll teach you about arrays.
Arrays?

That's right. Good idea.
Let's leave this program here.

If you think of a variable as a box to store something,

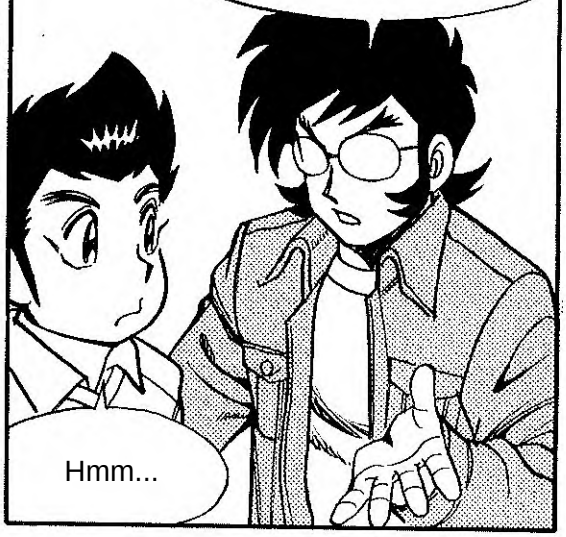
an array is a bunch of boxes lined up together.

← Variable

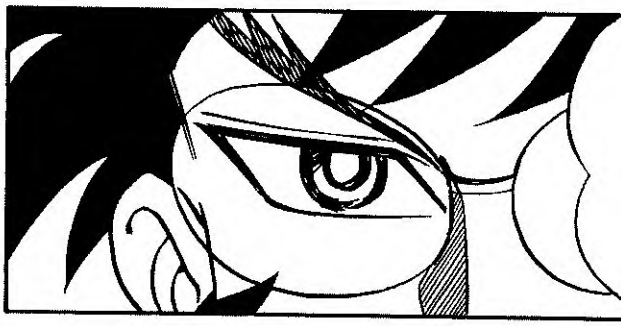
← Array

Arrays are technically array variables.

Normal variables can only hold one value or string.



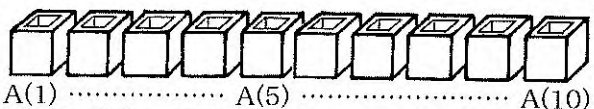
Hmm...



Variables don't need any command when setting up,

but Arrays use the DIM command when first declared.

The 10 in the parentheses after A determines how many boxes to prepare.

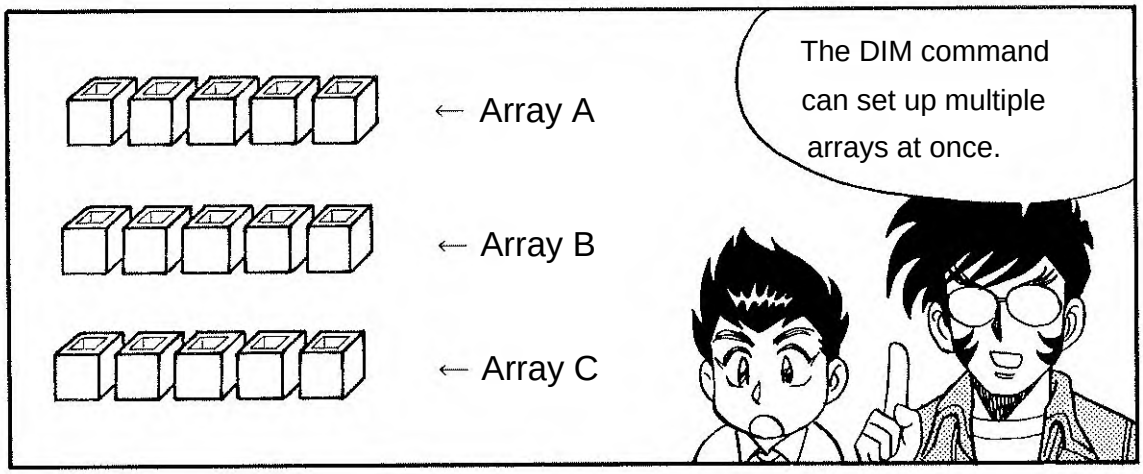


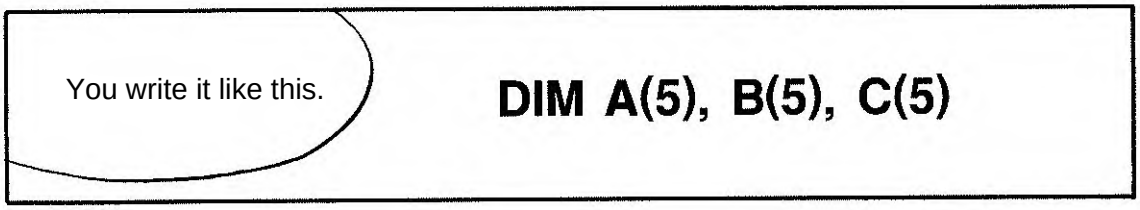
This means A(1)~A(10) are set up as variables.

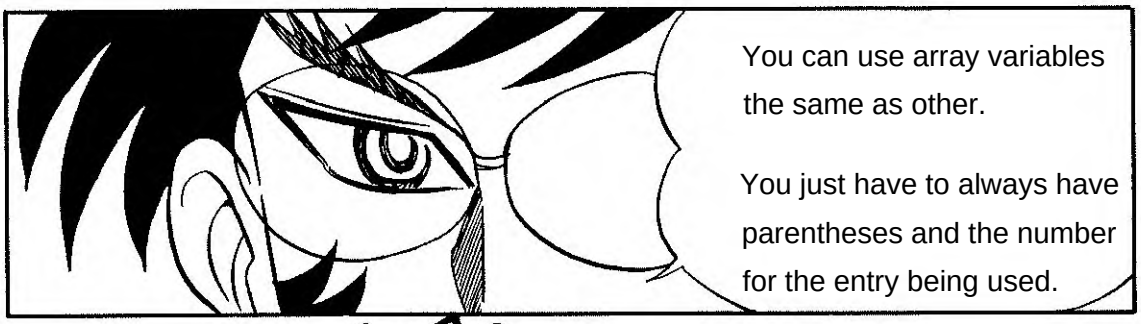
Like this.

DIM A(10)

This prepares an array called A.

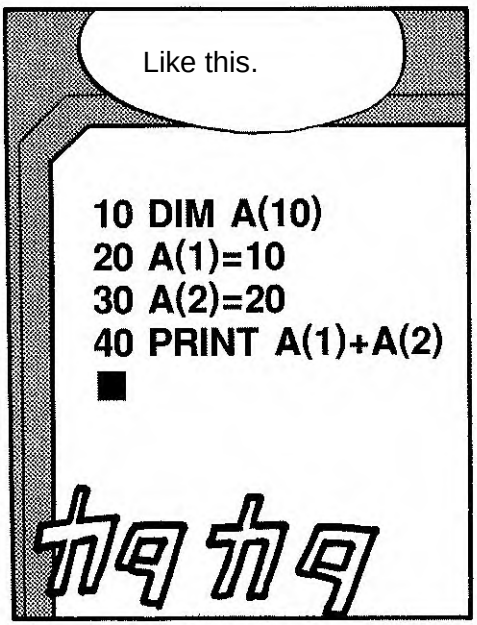

← Array A
← Array B
← Array C
The DIM command can set up multiple arrays at once.

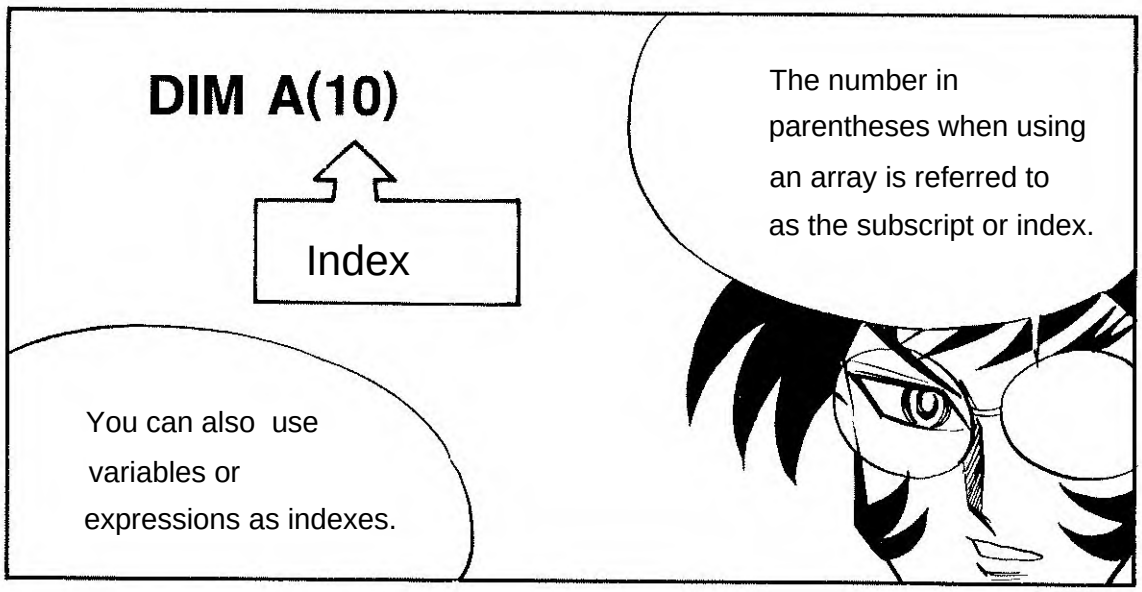

You write it like this.
DIM A(5), B(5), C(5)

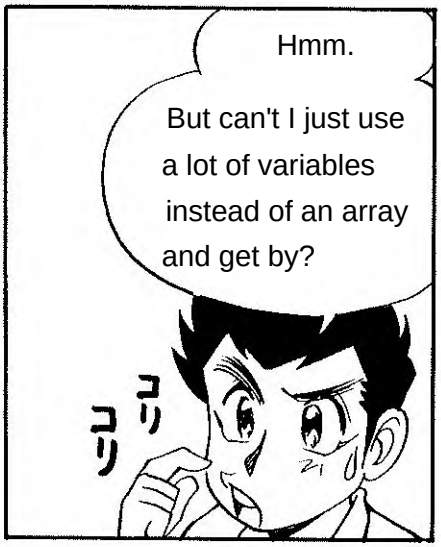

You can use array variables the same as other.
You just have to always have parentheses and the number for the entry being used.

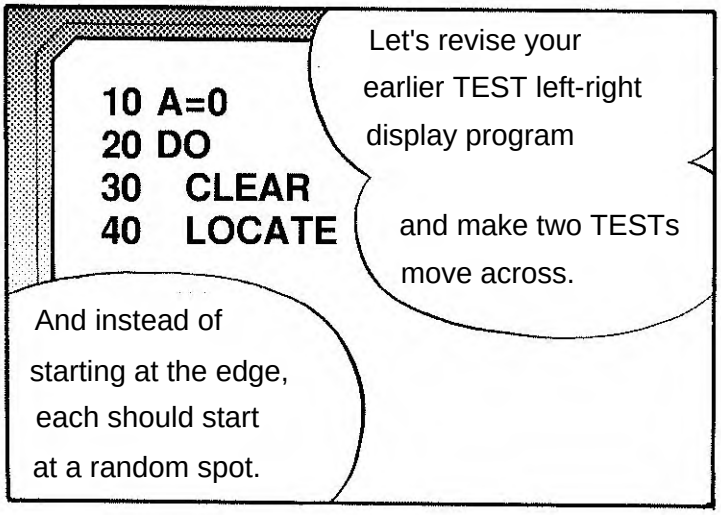
Let's try running this.
カチッ
Ent

RUN
実行を開始します。
30
実行を終了
■
ハ°ッ


Like this.
10 DIM A(10)
20 A(1)=10
30 A(2)=20
40 PRINT A(1)+A(2)
■
カタカタ


DIM A(10)
Index
The number in parentheses when using an array is referred to as the subscript or index.
You can also use variables or expressions as indexes.




10 A=0
20 DO
30 CLEAR
40 LOCATE
Let's revise your earlier TEST left-right display program
and make two TESTs move across.
And instead of starting at the edge, each should start at a random spot.

* Try yourself before looking at the answer.

Since I want to move two at once, I guess...

I could just duplicate all of this code.

Remaking starting at line 100 should work ok.

The previous program was like this.

```
LIST
10 A=0
20 DO
30   CLEAR
40   LOCATE A,0
50   PRINT "TEST"
60   A=A+1
70   FSYNC 60
80 LOOP UNTIL A=32
■
```

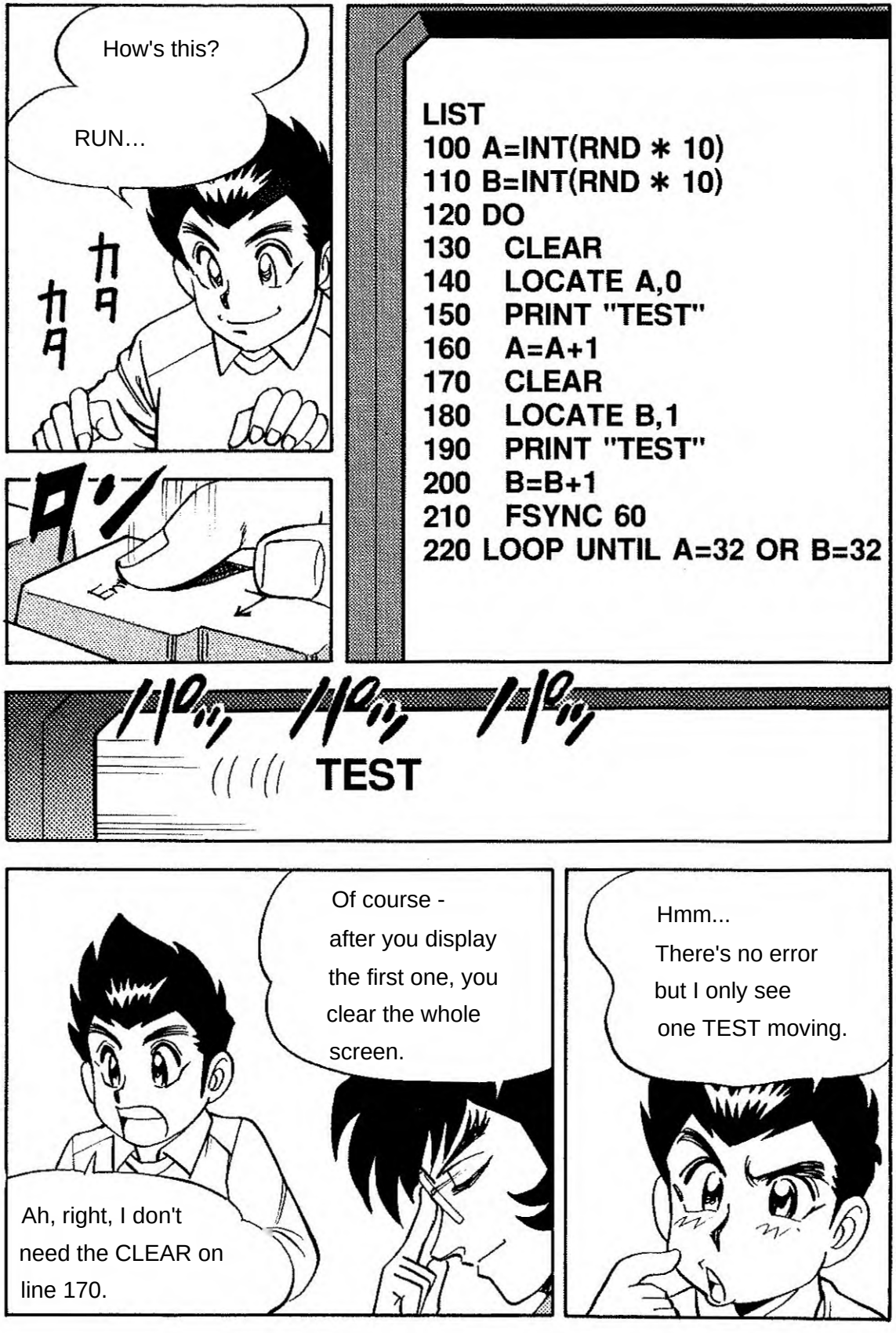
```
100 A=INT(RND * 10)
110 B=INT(RND * 10)
■
```

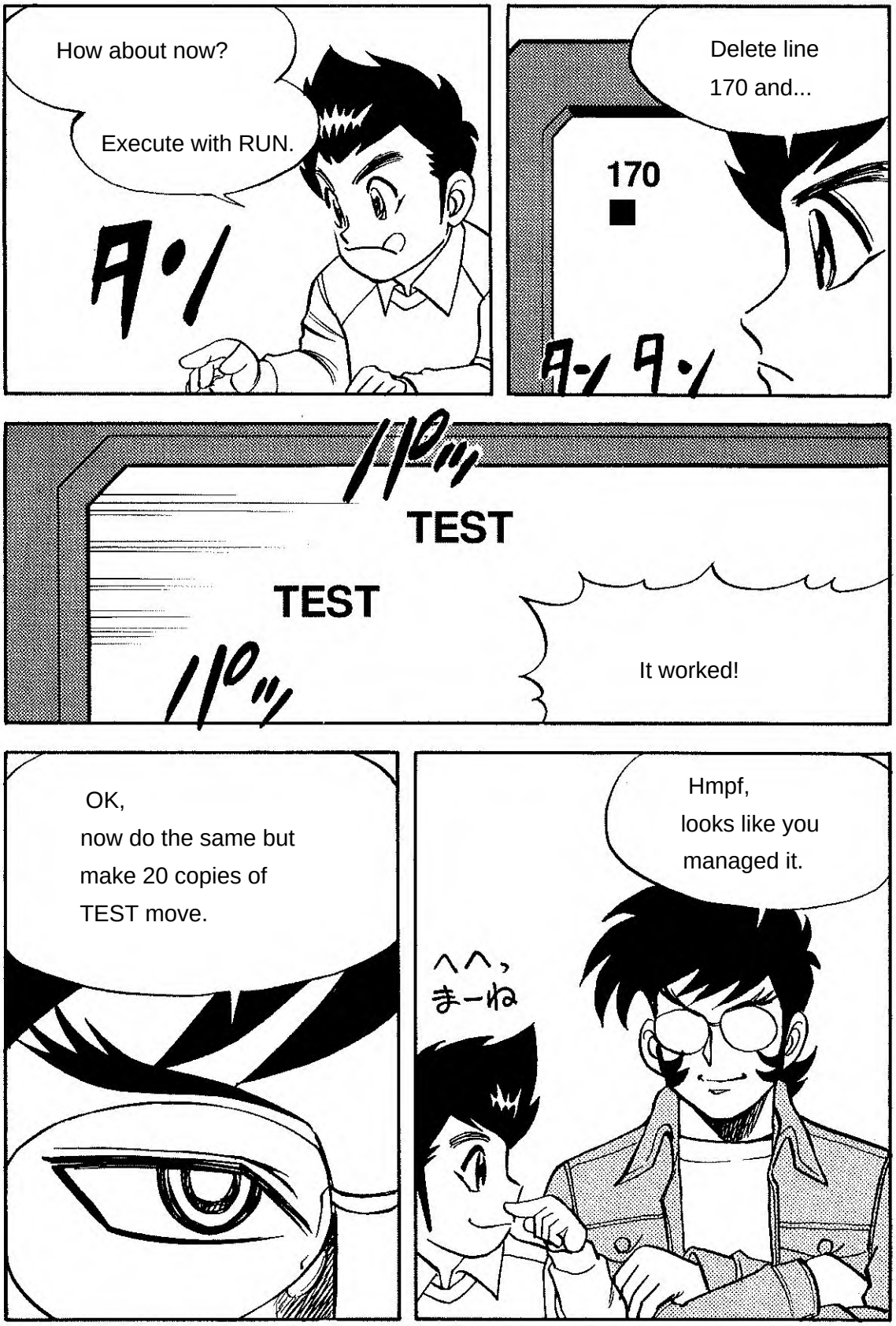
Use two variables

And set a random start position...

```
210   FSYNC 60
220 LOOP UNTIL A=32 OR B=32
```

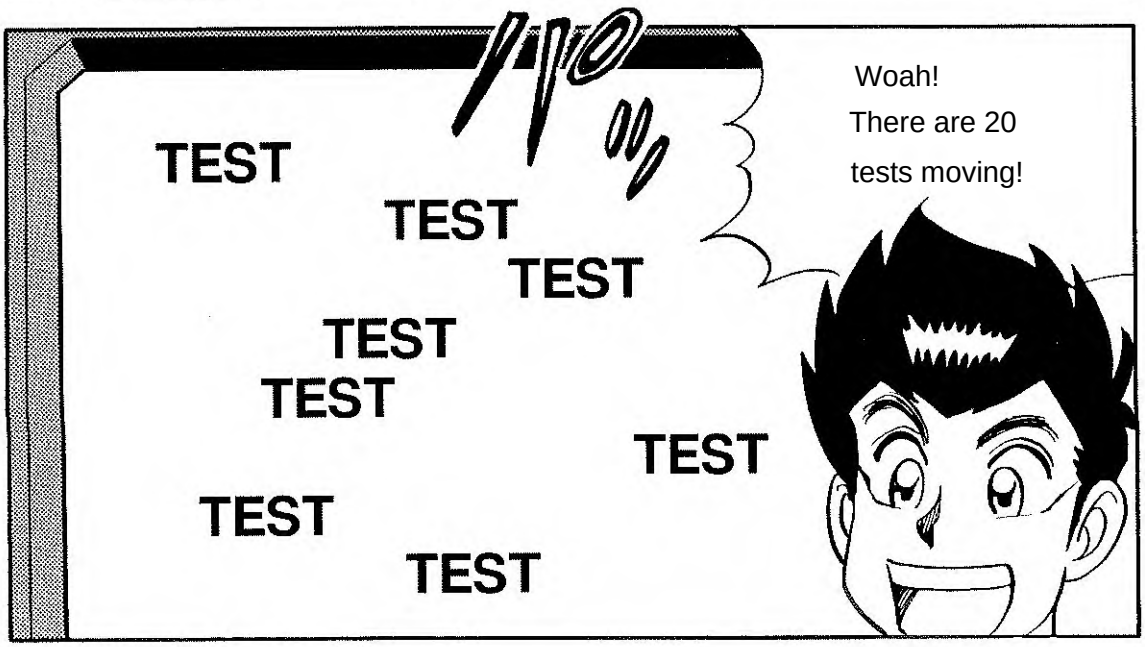
What should the loop end condition be? Maybe if either variable gets to 32...


How's this?
RUN...
カタ
カタ
LIST
100 A=INT(RND * 10)
110 B=INT(RND * 10)
120 DO
130 CLEAR
140 LOCATE A,0
150 PRINT "TEST"
160 A=A+1
170 CLEAR
180 LOCATE B,1
190 PRINT "TEST"
200 B=B+1
210 FSYNC 60
220 LOOP UNTIL A=32 OR B=32
TEST
Of course - after you display the first one, you clear the whole screen.
Ah, right, I don't need the CLEAR on line 170.
Hmm... There's no error but I only see one TEST moving.

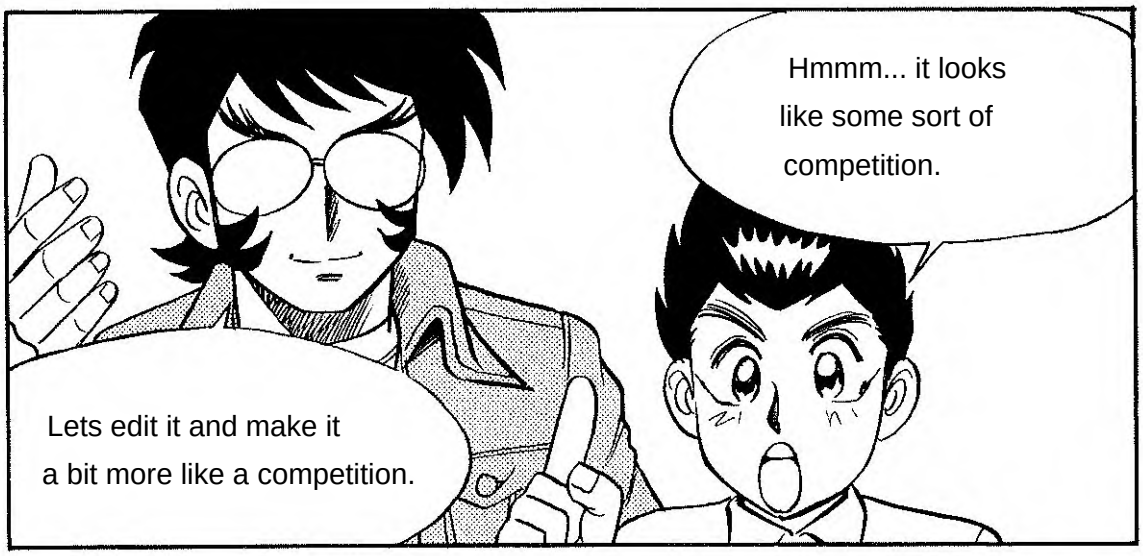

How about now?
Execute with RUN.
Delete line 170 and...
170
TEST
TEST
It worked!
OK, now do the same but make 20 copies of TEST move.
Hmpf, looks like you managed it.
へへ、まーね

Well, I'll set up 20 variables...
The program will get pretty long...
What? How will I be able to do that many?
Gah!! Writing the same program out 20 times is super tedious!
I'll try writing for you.
Give this a run.
ﾀﾀﾝ
That's right! This is where arrays are useful.

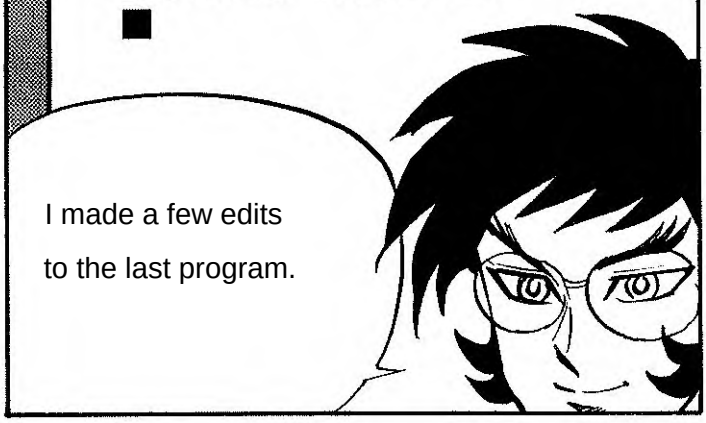
```
100 DIM X(20)
110 FOR Y=1 TO 20
120   X(Y)=INT(RND * 10)
130 NEXT Y
140 F=0
150 DO
160   CLEAR
170   FOR Y=1 TO 20
180     LOCATE X(Y),Y
190     PRINT "TEST"
200     X(Y)=X(Y)+1
210     IF X(Y)=32 THEN F=1
220   NEXT Y
230   FSYNC 60
240 LOOP UNTIL F=1
■
```

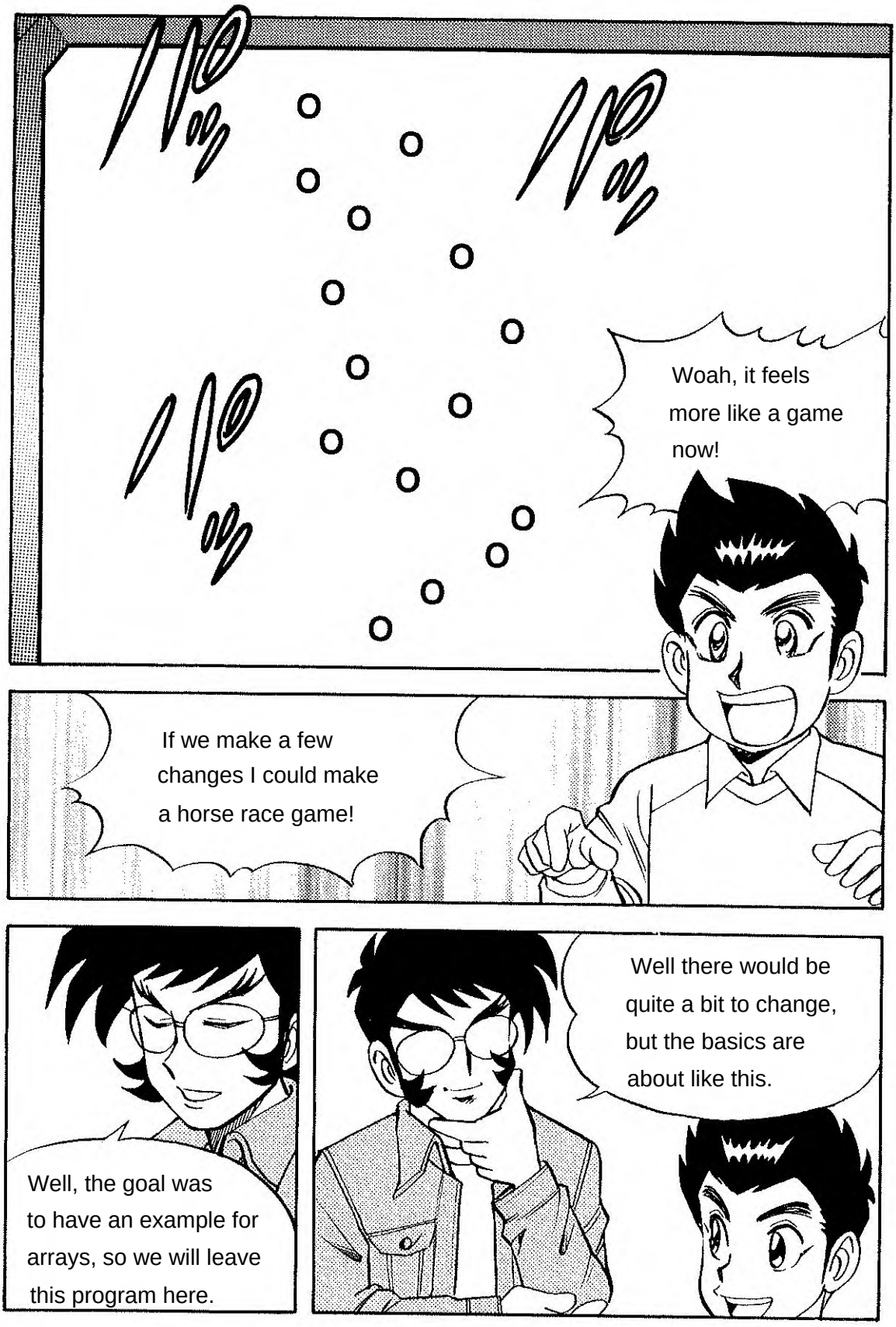


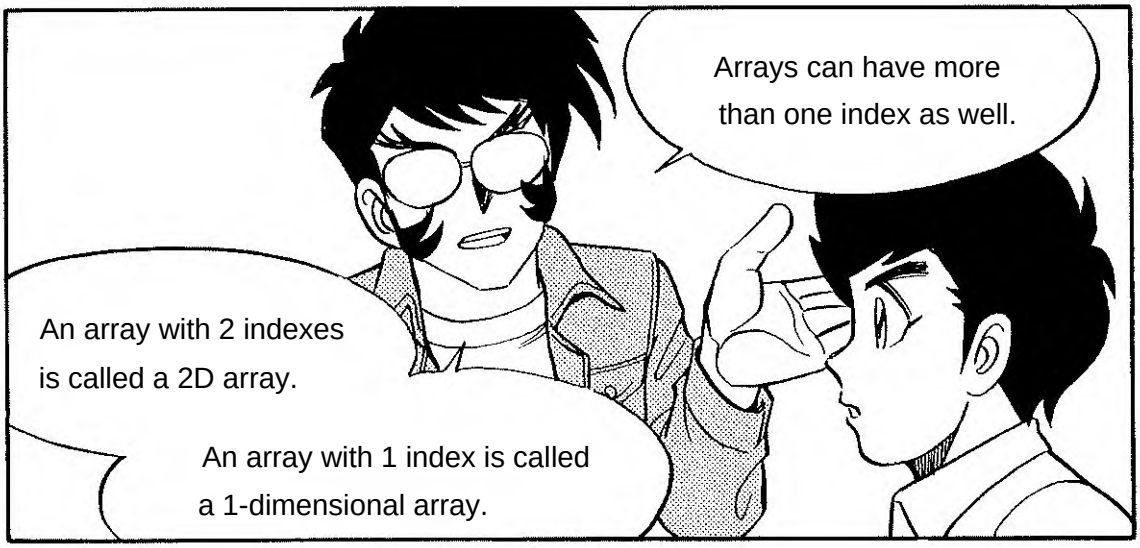
There are two places with FOR-NEXT loops.
Using Y as the index of the array, you can process 20 variables at once.
If any of them get to the edge of the screen, they set the variable F to 1 and that ends the loop.
110 FOR Y=
120 X(Y)=IN
130 NEXT Y
170 FOR Y=1
180 LOCAT
190 PRINT
And the program isn't much longer at all...
Using arrays, even doing 100 at once doesn't make the program longer.
By the way, looking at this screen, does anything come to mind?
TEST
TEST
TEST
TEST
Yeah, they're useful.
But it's impossible to do 100 lines of text, practically speaking.
You should see the use of arrays now, right?

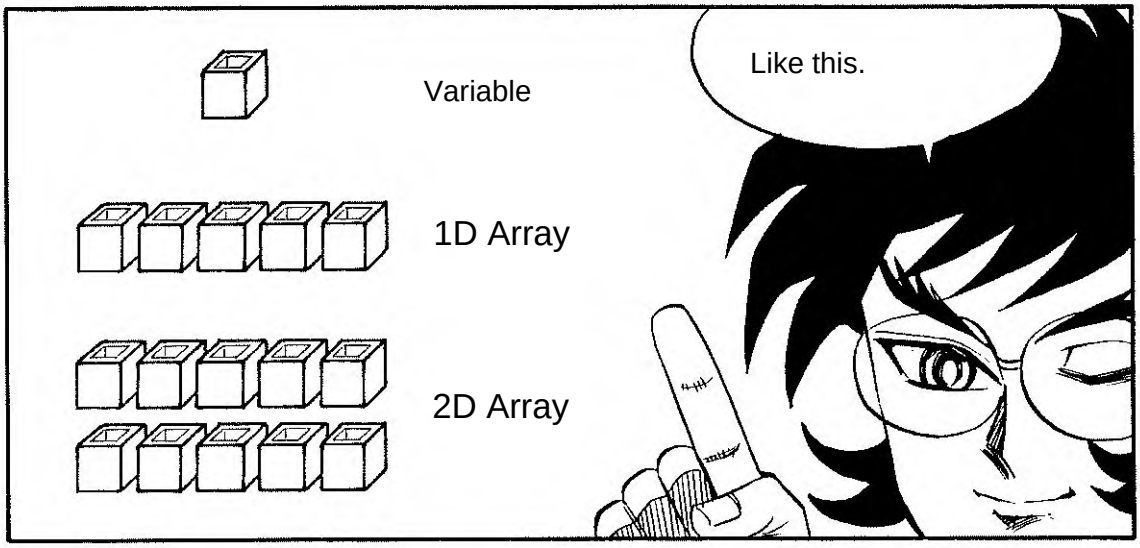


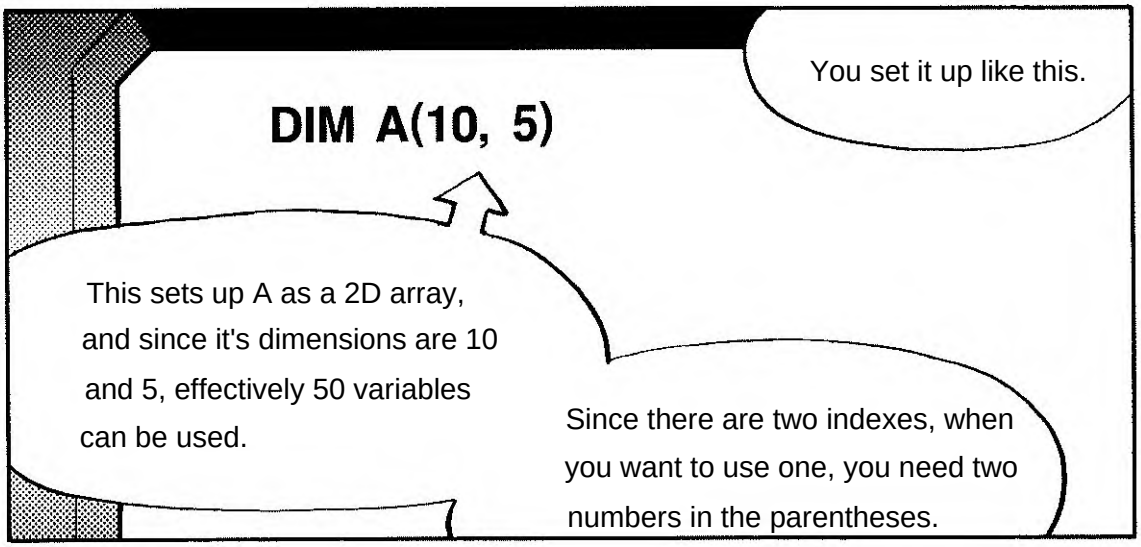
```
100 DIM X(20)
110 RANDOMIZE
120 F=0
130 DO
140   CLEAR
150   FOR Y=1 TO 20
160     LOCATE X(Y),Y
170     PRINT "O"
180     IF INT(RND * 3) > 1
 THEN X(Y)=X(Y)+1 (Line 180 continued)
190     IF X(Y)=32 THEN F=1
200   NEXT Y
210   FSYNC 60
220 LOOP UNTIL F=1
■
```

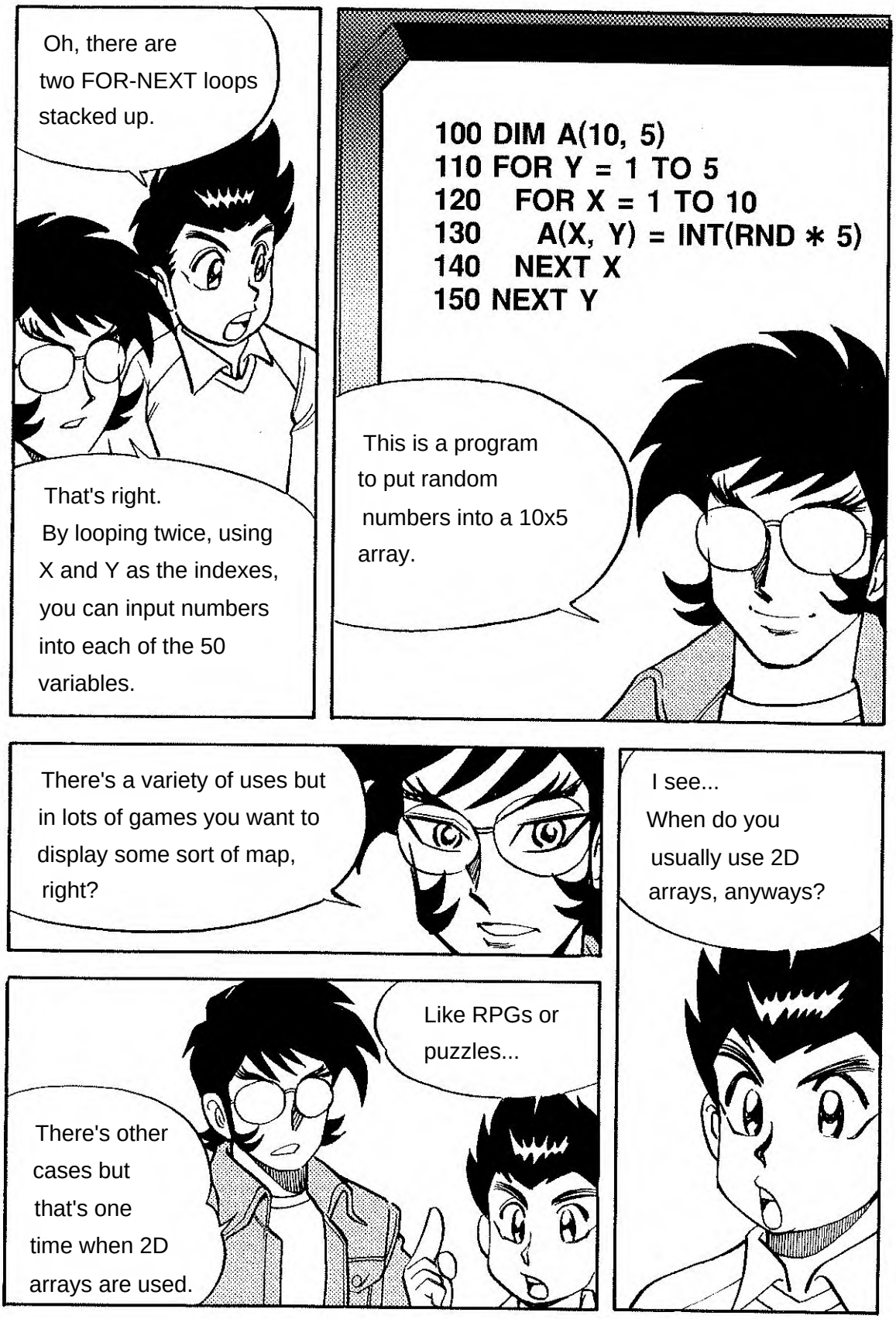


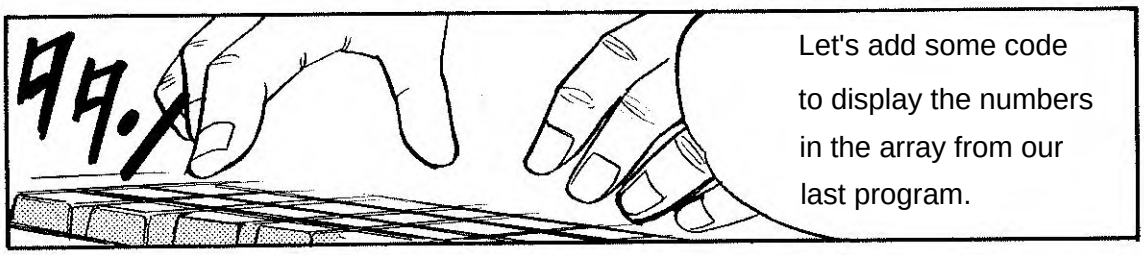

Woah, it feels more like a game now!
If we make a few changes I could make a horse race game!
Well, the goal was to have an example for arrays, so we will leave this program here.
Well there would be quite a bit to change, but the basics are about like this.

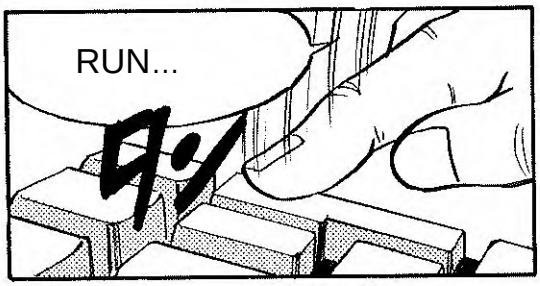

Arrays can have more than one index as well.
An array with 2 indexes is called a 2D array.
An array with 1 index is called a 1-dimensional array.


Variable
1D Array
2D Array
Like this.


You set it up like this.
DIM A(10, 5)
This sets up A as a 2D array, and since it's dimensions are 10 and 5, effectively 50 variables can be used.
Since there are two indexes, when you want to use one, you need two numbers in the parentheses.

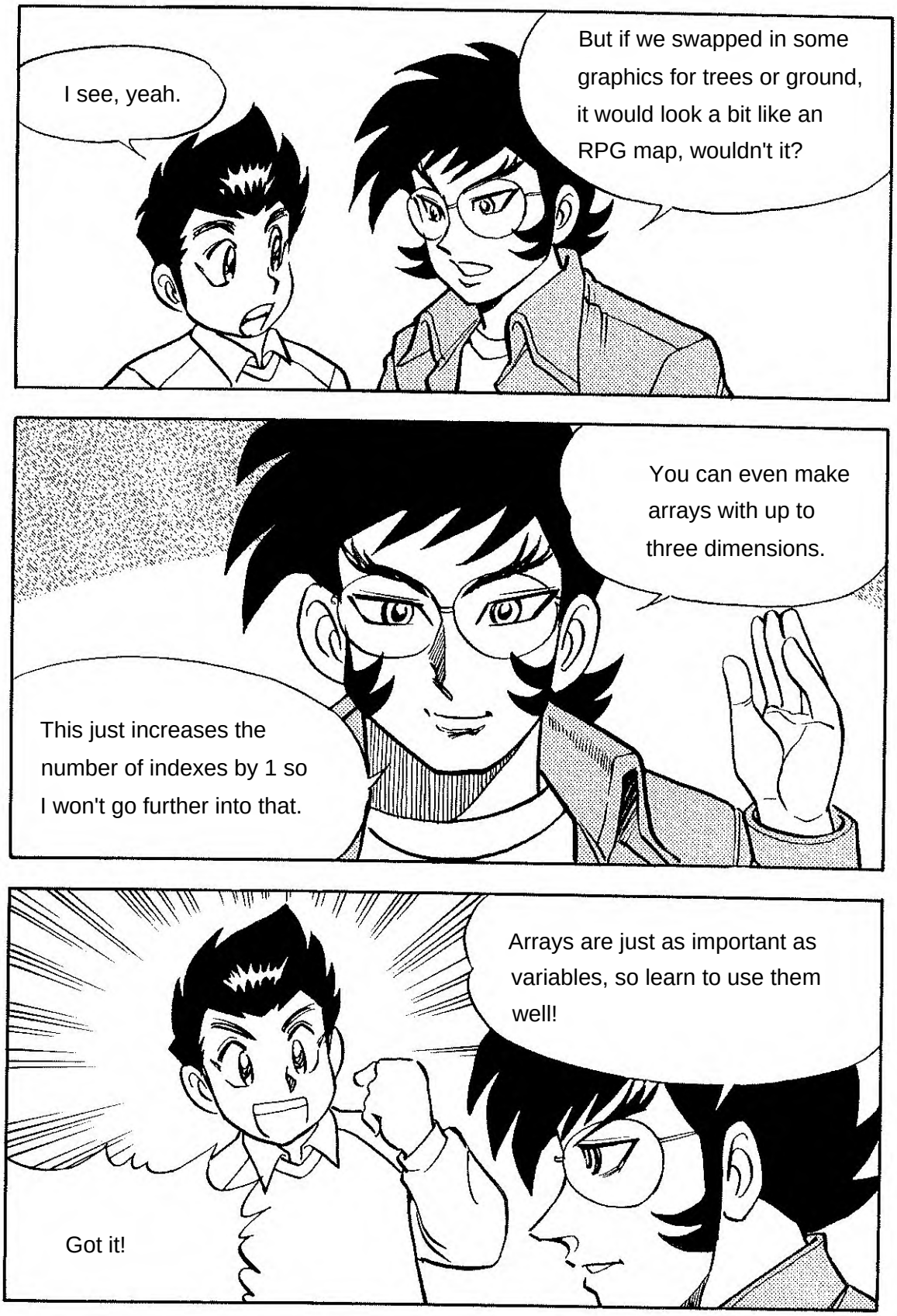

Oh, there are two FOR-NEXT loops stacked up.
That's right. By looping twice, using X and Y as the indexes, you can input numbers into each of the 50 variables.
100 DIM A(10, 5)
110 FOR Y = 1 TO 5
120 FOR X = 1 TO 10
130 A(X, Y) = INT(RND * 5)
140 NEXT X
150 NEXT Y
This is a program to put random numbers into a 10x5 array.
There's a variety of uses but in lots of games you want to display some sort of map, right?
I see... When do you usually use 2D arrays, anyways?
Like RPGs or puzzles...
There's other cases but that's one time when 2D arrays are used.

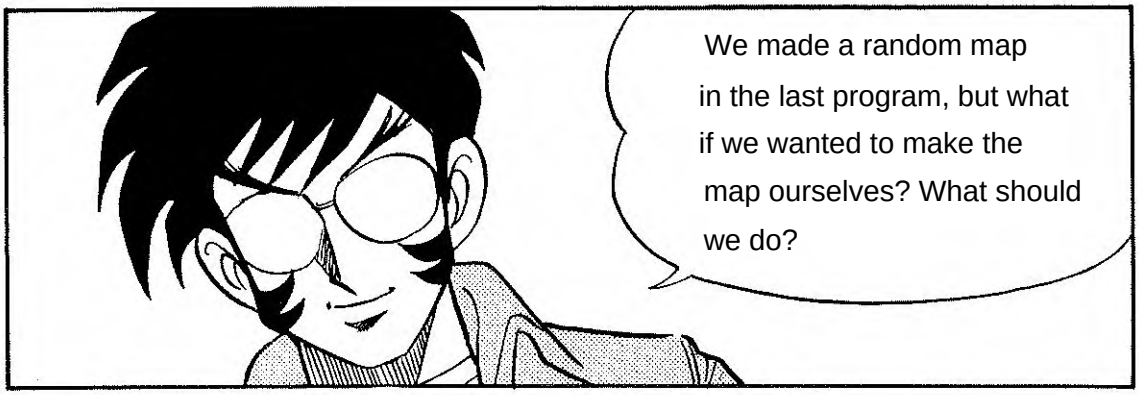




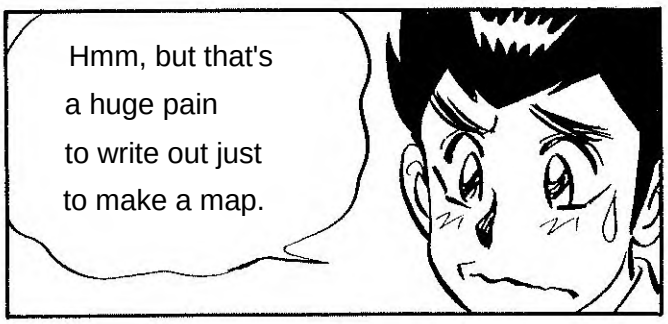

```
100 DIM A(10, 5)
110 FOR Y = 1 TO 5
120   FOR X = 1 TO 10
130     A(X, Y) = INT(RND * 5)
140   NEXT X
150 NEXT Y
160 CLEAR
170 FOR Y = 1 TO 5
180   FOR X = 1 TO 10
190     LOCATE X, Y
200     PRINT A(X, Y)
210   NEXT X
220 NEXT Y
■
```

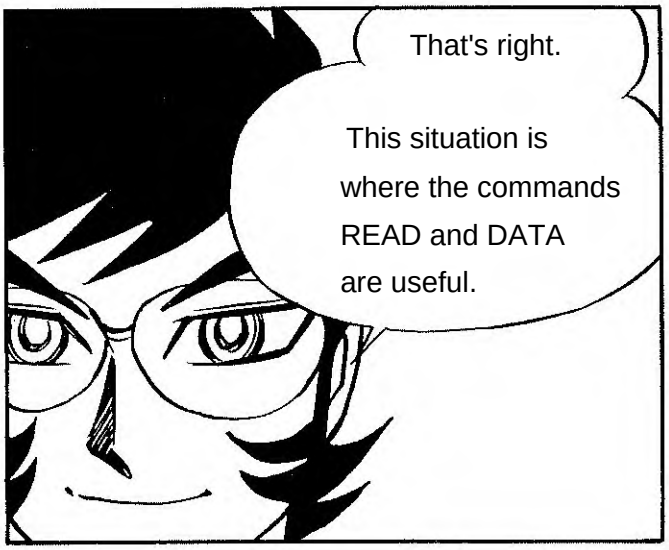
```
2012403120
0111423323
4031142332
4423341341
1244230414
```

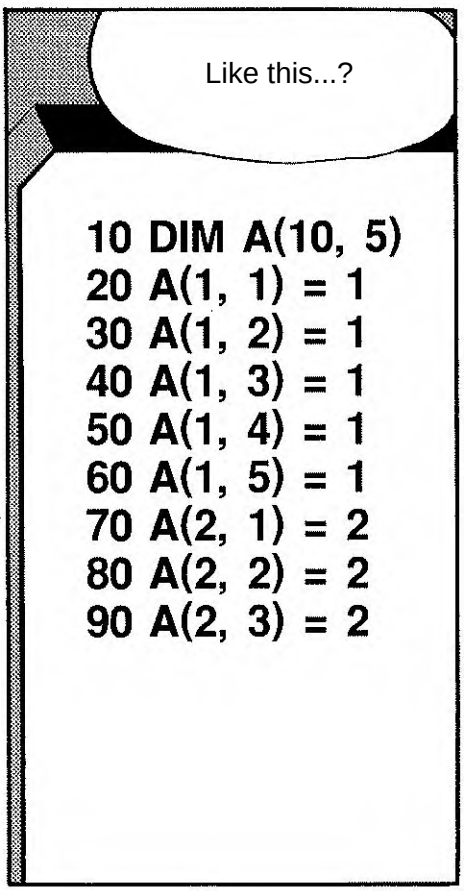

I see, yeah.
But if we swapped in some graphics for trees or ground, it would look a bit like an RPG map, wouldn't it?
You can even make arrays with up to three dimensions.
This just increases the number of indexes by 1 so I won't go further into that.
Arrays are just as important as variables, so learn to use them well!
Got it!

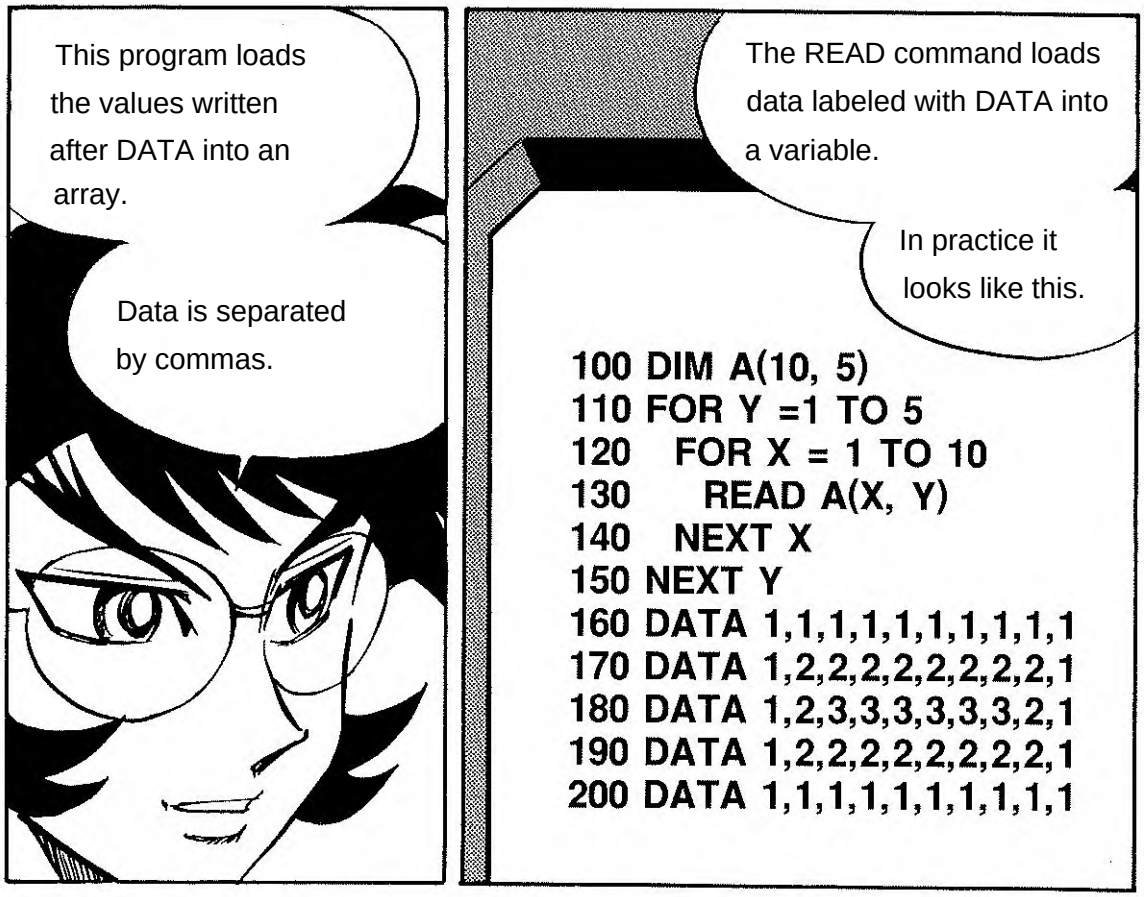

We made a random map in the last program, but what if we wanted to make the map ourselves? What should we do?

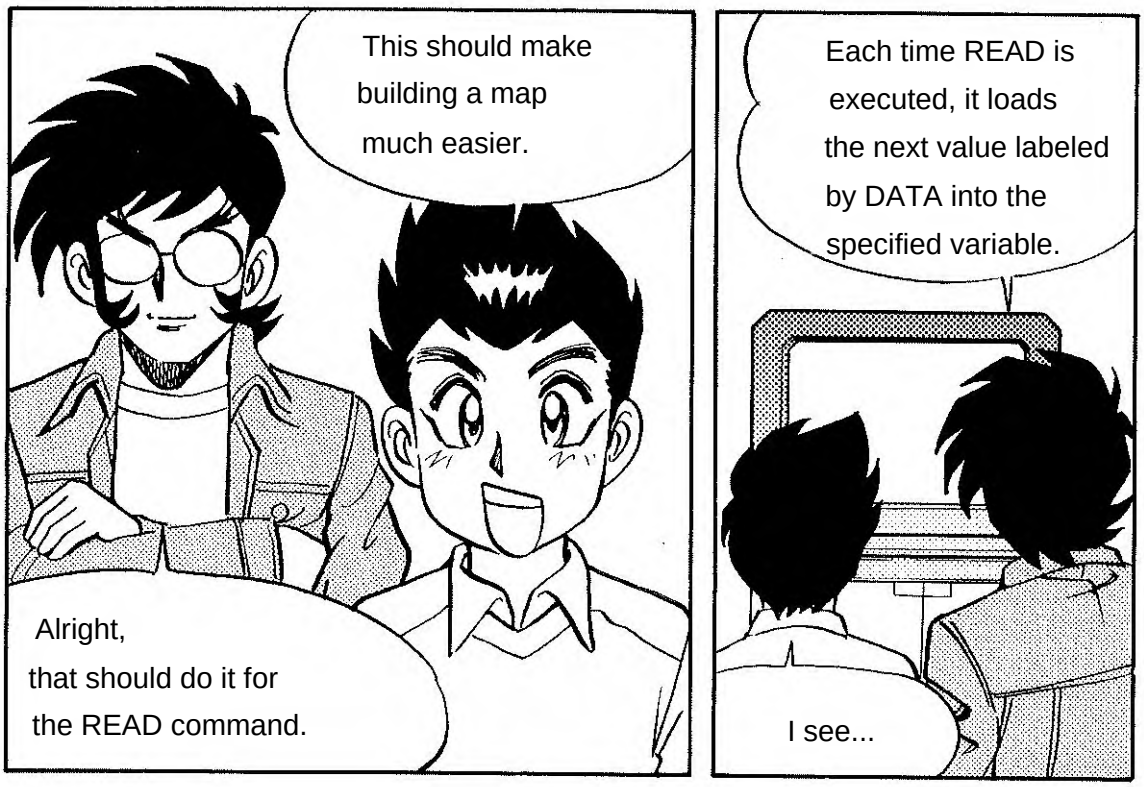
Hmmm... I gues we could just put numbers into the array...


Hmm, but that's a huge pain to write out just to make a map.


That's right.
This situation is where the commands READ and DATA are useful.


Like this...?
10 DIM A(10, 5)
20 A(1, 1) = 1
30 A(1, 2) = 1
40 A(1, 3) = 1
50 A(1, 4) = 1
60 A(1, 5) = 1
70 A(2, 1) = 2
80 A(2, 2) = 2
90 A(2, 3) = 2


This program loads the values written after DATA into an array.
Data is separated by commas.
The READ command loads data labeled with DATA into a variable.
In practice it looks like this.
100 DIM A(10, 5)
110 FOR Y =1 TO 5
120 FOR X = 1 TO 10
130 READ A(X, Y)
140 NEXT X
150 NEXT Y
160 DATA 1,1,1,1,1,1,1,1,1,1
170 DATA 1,2,2,2,2,2,2,2,2,1
180 DATA 1,2,3,3,3,3,3,3,2,1
190 DATA 1,2,2,2,2,2,2,2,2,1
200 DATA 1,1,1,1,1,1,1,1,1,1


This should make building a map much easier.
Alright, that should do it for the READ command.
Each time READ is executed, it loads the next value labeled by DATA into the specified variable.
I see...

When making a game, you often need to use the same program in multiple places.

In that situation, by using a subroutine, you can use the same program as many times as you want.

Subprograms

Next are subprograms (subroutines).

Subprogram?

Subprograms are declared with the SUB command.

カタ カタ

Defined subprograms are run using the CALL.

Like this.

```
10 CALL SP
20 CALL SP
30 SUB SP
40   PRINT "TEST"
50 END SUB
```

```
10 CALL SP
20 CALL SP
```

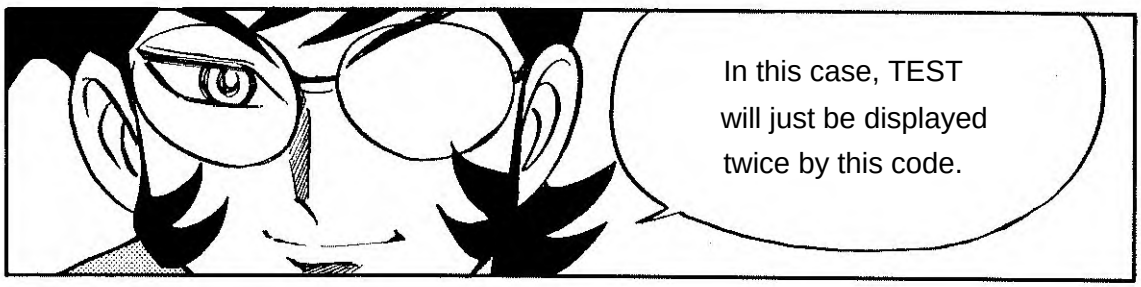
This subprogram is called up by CALL on lines 10 and 20.

```
30 SUB SP
40   PRINT "TEST"
50 END SUB
```

When CALL runs a subroutine, it starts from the line with SUB and executes code until END SUB, then returns to where CALL happened.

On line 30, the SUB command defines a sub-routine with the name SP, and the program is defined until END SUB on line 50.

That said, the code inside is just to display TEST



In this case, TEST will just be displayed twice by this code.

When executing a program, if you get to a SUB line without going through CALL, it will skip to END SUB.

If you don't run it with CALL, nothing inside a SUB definition will ever run.

Oh, I see.

Ok...

But after the two CALL commands once the code gets to line 30, won't it print out a third copy of test?

To do this, put parentheses and a variable after the name during the SUB declaration.

```
SUB SP(A)
```

```
CALL SP(100)
```

For the call command, you put the input in parentheses after the subprogram name.

You can also give subprograms inputs like functions.

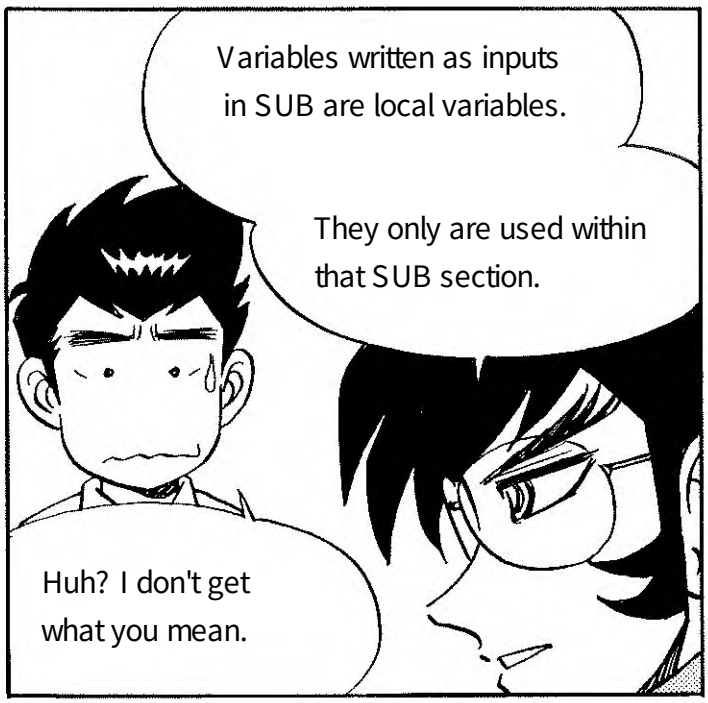
In practice it looks like this.

```
10 CALL SP(100)
20 CALL SP(200)
30 SUB SP(A)
40   PRINT "TEST", A
50 END SUB
```

Hmm... okay.

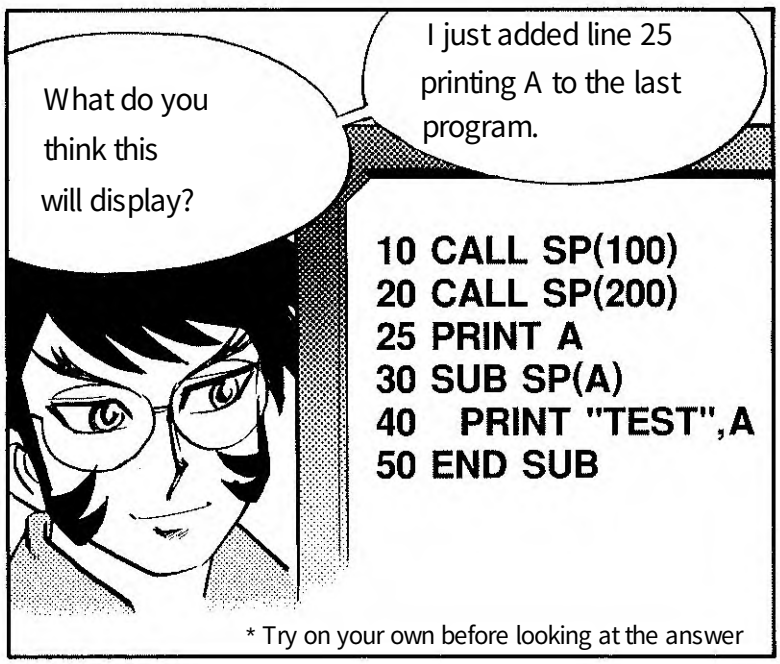
In this example, the variable A is taken as input and printed after TEST.

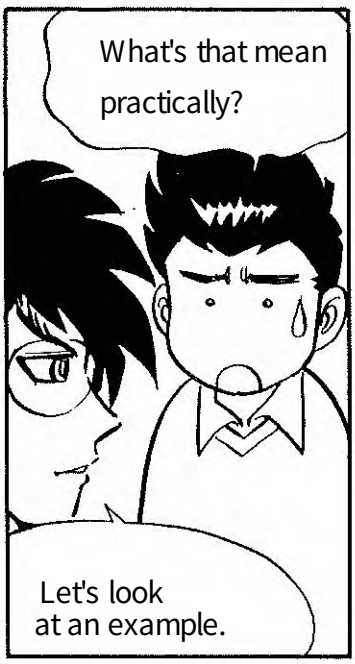
The call command is passing 100 and 200 as inputs on 10 and 20.


Variables written as inputs in SUB are local variables.
They only are used within that SUB section.
Huh? I don't get what you mean.

In this case, the variable we wrote in SUB differs from the variables we have used so far.
Huh?
In what way?

That is, once END SUB is reached, the variable is discarded.


What do you think this will display?
I just added line 25 printing A to the last program.
10 CALL SP(100)
20 CALL SP(200)
25 PRINT A
30 SUB SP(A)
40 PRINT "TEST",A
50 END SUB
* Try on your own before looking at the answer


What's that mean practically?
Let's look at an example.

Ok, so on line 10 SP is given the value 100 and runs...

```
10 CALL SP(100)
```

```
30 SUB SP(A)

40 PRINT "TEST", A
```

And in the SUB, the variable A has the value 100 and is printed after TEST, so it will show "TEST 100".

Then it goes back to CALL when it gets to END SUB.

```
50 END SUB
```

```
20 CALL SP(200)
```

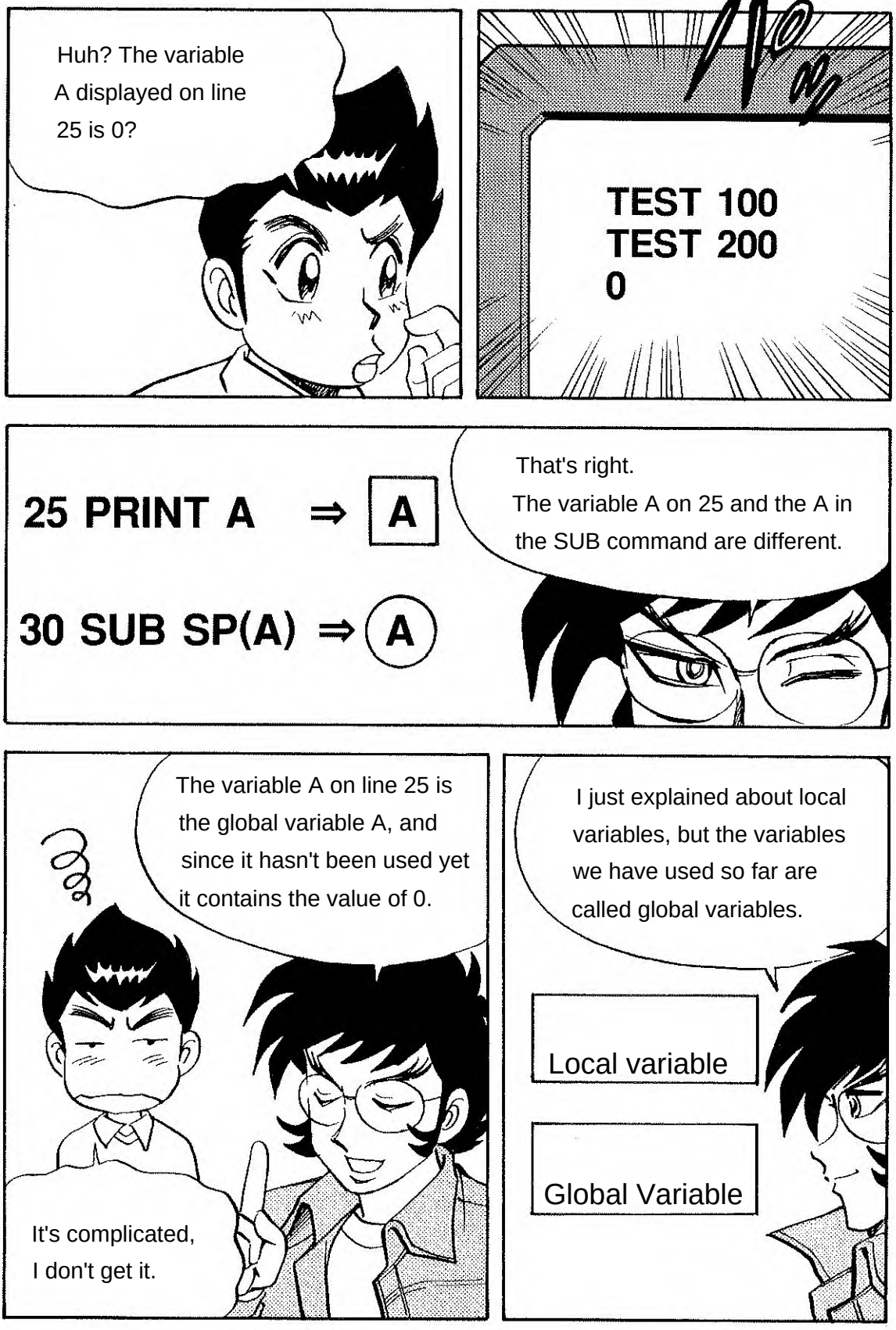
Then on line 20 the same thing but with 200 happens, so "TEST 200" will show.

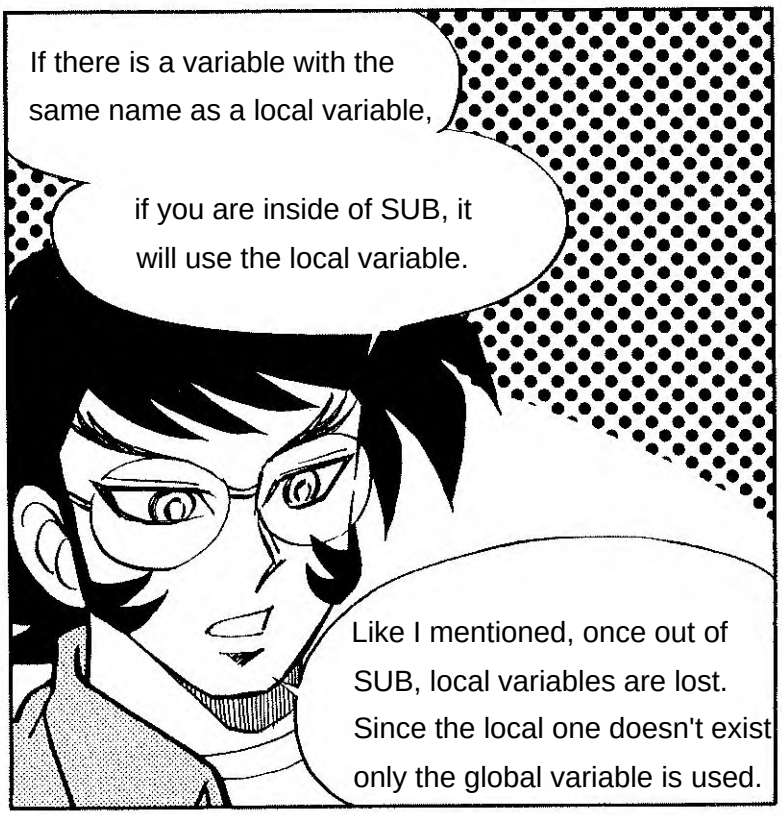
On line 25 it displays A, and A was just set to 200 when it ran before, so I guess it should print 200?

```
25 PRINT A
```

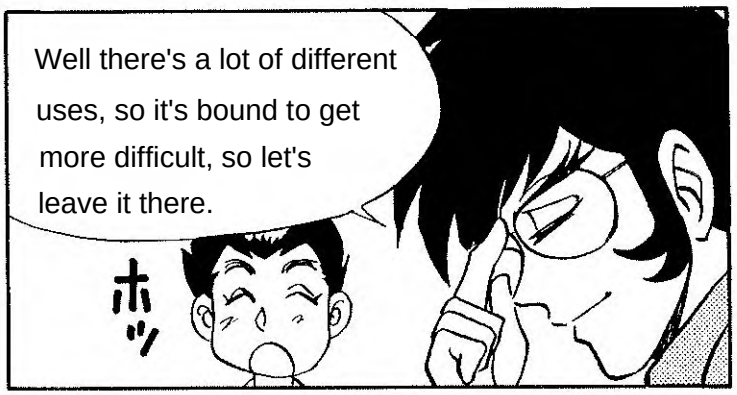
So it should show something like this?

```
TEST 100
TEST 200
200
```


Huh? The variable A displayed on line 25 is 0?
TEST 100
TEST 200
0
25 PRINT A ⇒ A
30 SUB SP(A) ⇒ A
That's right.
The variable A on 25 and the A in the SUB command are different.
The variable A on line 25 is the global variable A, and since it hasn't been used yet it contains the value of 0.
It's complicated, I don't get it.
I just explained about local variables, but the variables we have used so far are called global variables.
Local variable
Global Variable


If there is a variable with the same name as a local variable,
if you are inside of SUB, it will use the local variable.
Like I mentioned, once out of SUB, local variables are lost. Since the local one doesn't exist only the global variable is used.

So basically if there are two variables with the same name, how can you tell which will be used?


Well there's a lot of different uses, so it's bound to get more difficult, so let's leave it there.
ホッ

Umm...
It's still confusing!
Well, this subprogram stuff is indeed a bit complicated...

But I'll teach one more command.

It looks like EXIT DO or EXIT FOR, so maybe it is to get out of a SUB program early?
Correct.
The EXIT SUB command.
You can probably guess its use from the name.
EXIT SUB lets you get out of a subprogram early, and it returns to where the CALL command occurred.
It's simple so I'll leave it at that.
That much I can manage.
With that we are done with most basic commands.
Alright! Will do!
Make sure to review!

Column 3 Subprograms and References

In addition to numbers and strings, variables can be passed as arguments to subprograms.

There are some differences between passing values such as numbers or strings and passing variables as arguments.

This section explains the differences in processing between passing values and passing variables as arguments.

Passing as Value and as Reference

♦Pass by value

•Pass a value, such as a number or a string, to a subprogram. This is called "passing a value."

The passed value is assigned to the argument of the subprogram declared in the SUB line. This argument is called a dummy argument.

A dummy argument is a variable (local variable) that is valid only within a subprogram and is discarded when the subprogram ends.

Example:

```
100 X=10
110 CALL ADD(10)
120 PRINT X
130 SUB ADD(X)
140  X=X+5
150 END SUB
```

<Execution result>

```
10
```

Here, the value 10 is passed as an argument to the subprogram ADD. The variable X used in this subprogram acts as a local variable that is valid only within the subprogram, and is treated as a different variable from the variable X used outside. Therefore, the value 10 assigned to variable X on line 100 will continue to line 120 and display the value 10 on line 120.

♦Pass by reference

When you pass a variable to a subprogram, the subprogram treats it as a reference. This is called passing by reference.

Assignments to arguments of a subprogram declared in a SUB line are assignments to arguments passed by reference.

Example:

```
100 X=10
110 CALL ADD(X)
120 PRINT X
130 SUB ADD(A)
140  A=A+5
150 END SUB
```

<Execution result>

```
15
```

Here, variable X is passed to subprogram ADD by reference. Parameter A becomes a reference to variable X, and the assignment to parameter A in line 140 becomes an assignment to variable X. As a result, the value of variable X becomes 15.

Benefits of References

♦By using a reference, the contents of the referenced variable can be changed within the subprogram.

Column 3 Subprograms and Reference 副プログラムと参照

Passing variable values

•If you want to pass the value of a variable to an argument, writing the variable as is in the argument will result in passing it by reference. To pass the value of the variable to the argument instead of passing the variable by reference, either:

--- Surround variables with ( ).

---Write a formula that does not affect the value of the argument, such as adding 0 to a variable or multiplying it by 1.

Example:

```
100 X=10
110 CALL ADD((X))
120 PRINT X
130 SUB ADD(A)
140  A=A+5
150 END SUB

<Execution Result>
10
```

Here, the value of variable X is passed as an argument to the subprogram ADD. By enclosing the variable in parentheses () like this, the value of variable X, 10, is passed. Other ways of writing this include CALL ADD(X+0) and CALL ADD(X*1). In either case, the argument is an "expression," and the resulting "value" is passed.

Reference Notes

♦Local variables

•Dummy arguments used in a subprogram act as local variables.

Local variables are valid only within a subprogram and are destroyed when the subprogram ends.

A local variable is used within a subprogram even if a variable with the same name exists outside the subprogram. Therefore, there will be two variables with the same name but different uses, so be careful not to confuse them.

•Local variables cannot be used outside the subprogram.

If you use a variable with the same name as a local variable outside a subprogram, it becomes the new variable.

♦Variables in subprograms

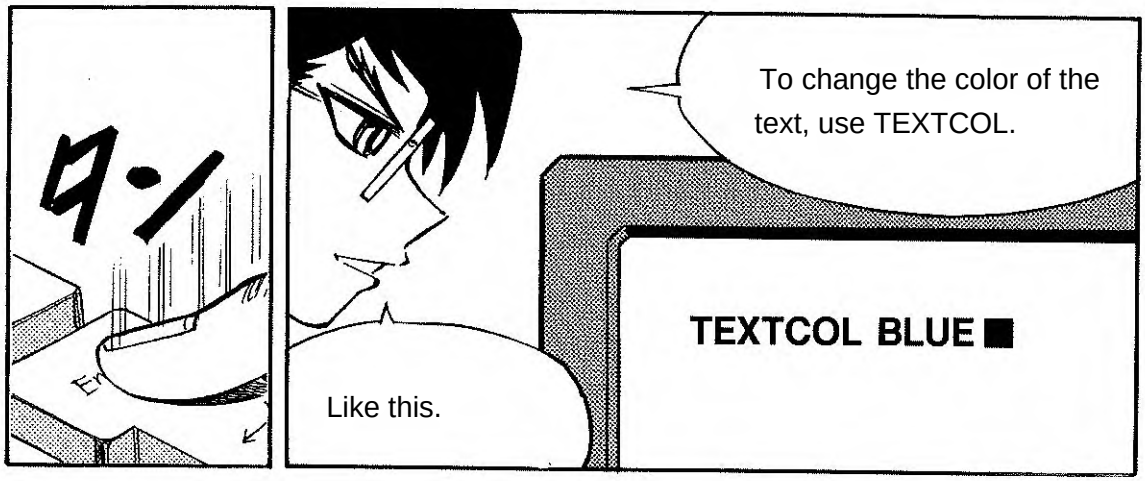
•Variables used in a subprogram, other than dummy arguments, are global variables.

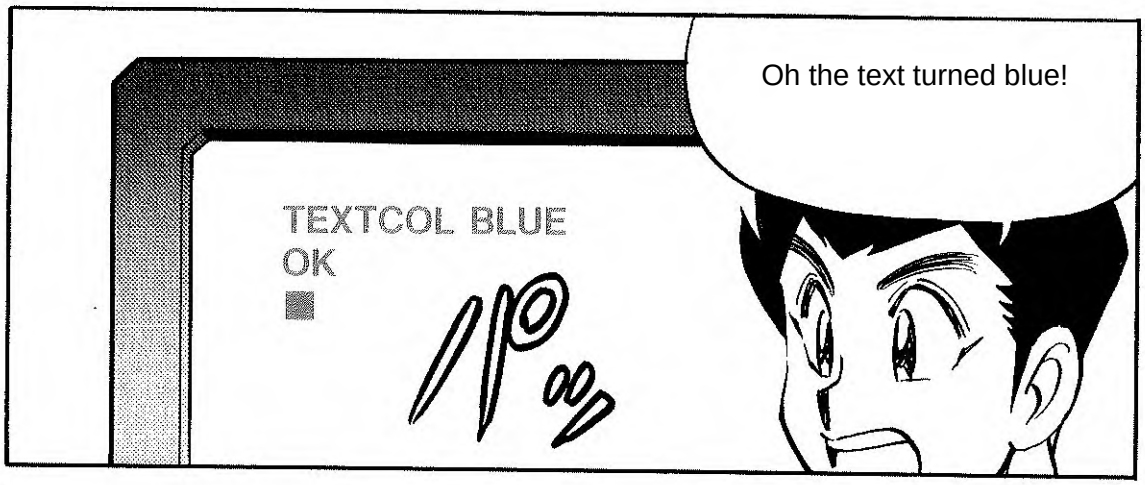
A global variable is a variable that can be used not only within the subprogram but also from outside it.

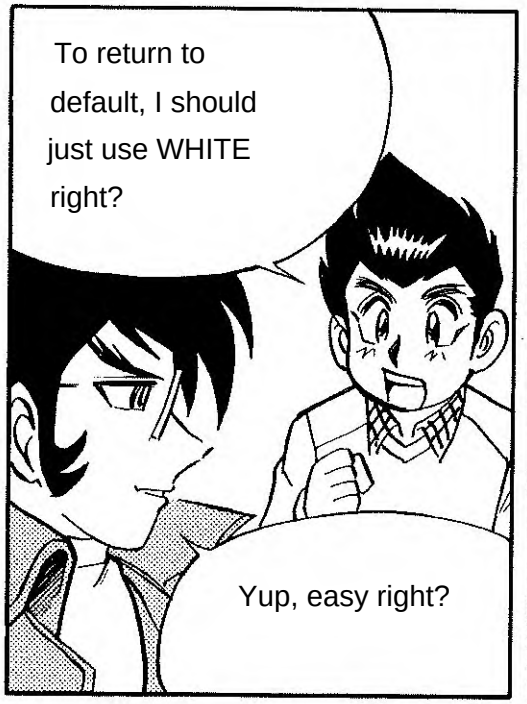
If you want to use it only within a subprogram, be careful not to use a variable name that is the same as a variable used outside the subprogram.

Translator's addition - On this last point, be careful with loop indexes, as a CALL to a subprogram from within a loop using an index, where that subprogram happens to have a loop with the same index variable, when the subprogram ends, the execution may return with a change to the index mid-stream.

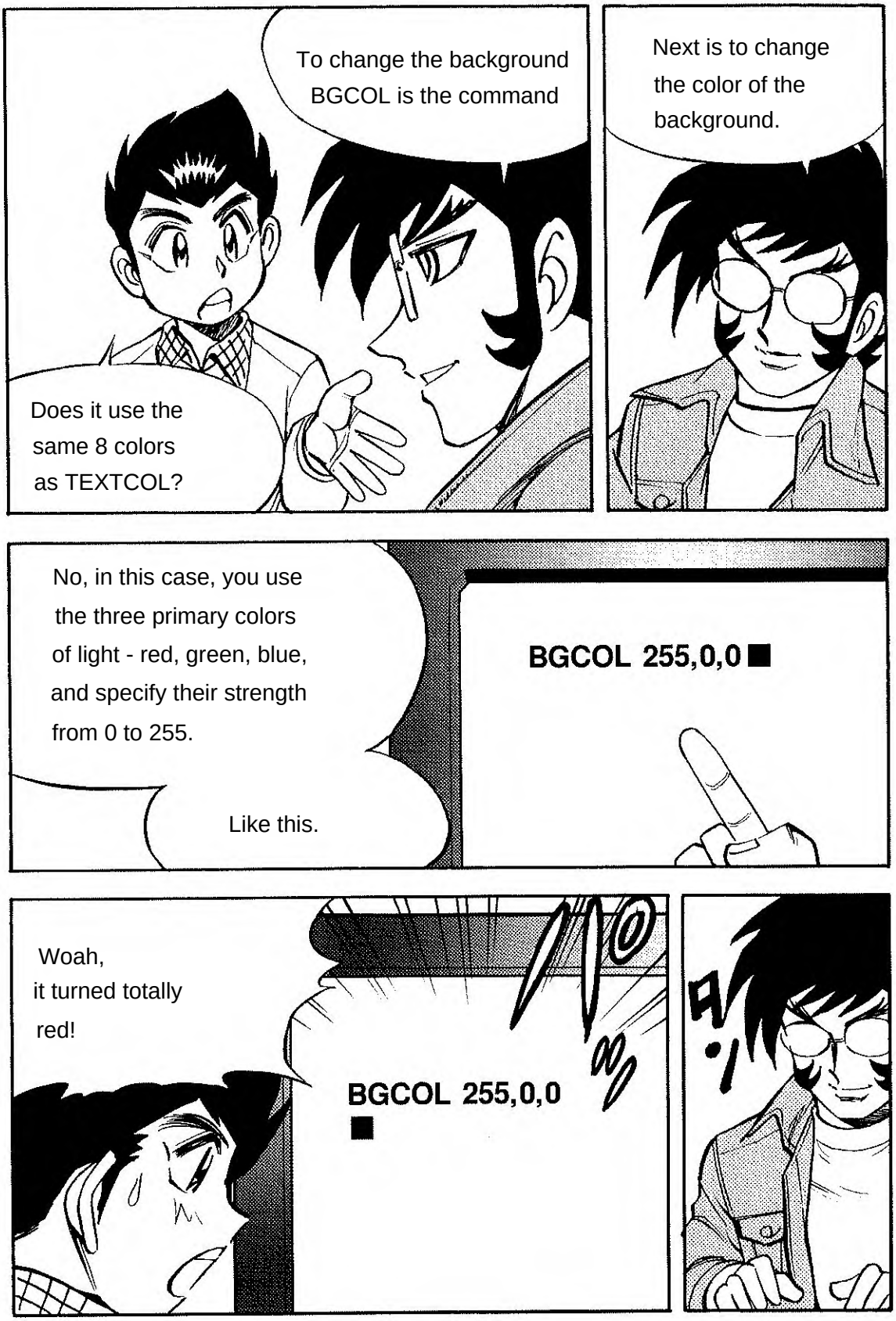
The challenge of game creation


To change the color of the text, use TEXTCOL.
TEXTCOL BLUE■
Like this.

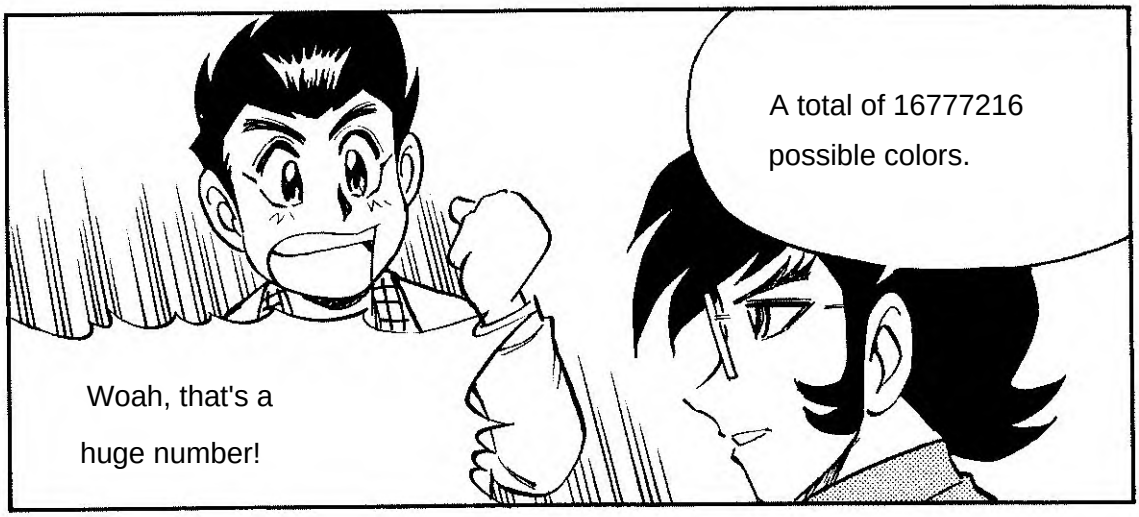

Oh the text turned blue!
TEXTCOL BLUE
OK
■

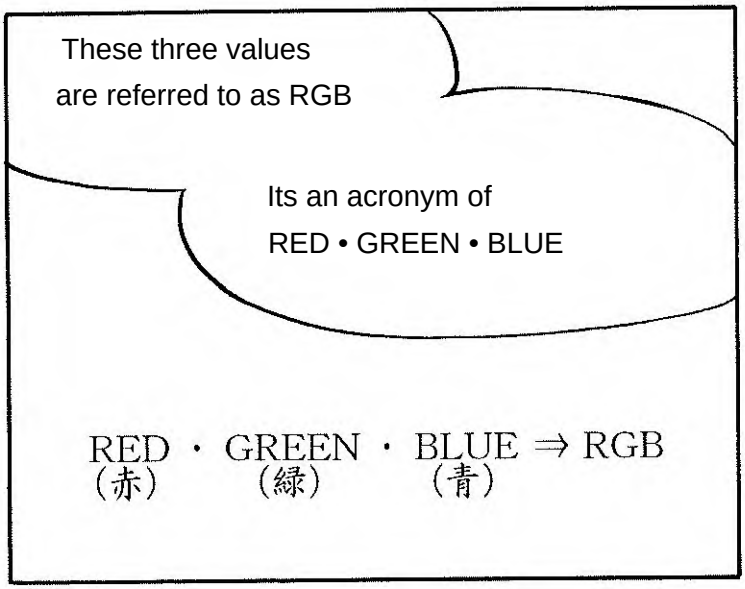

To return to default, I should just use WHITE right?
Yup, easy right?

Text can use these colors.
BLUE 青色
RED 赤色
GREEN 緑色
CYAN 水色
PURPLE 紫色
YELLOW 黄色
WHITE 白色
BLACK 黒色

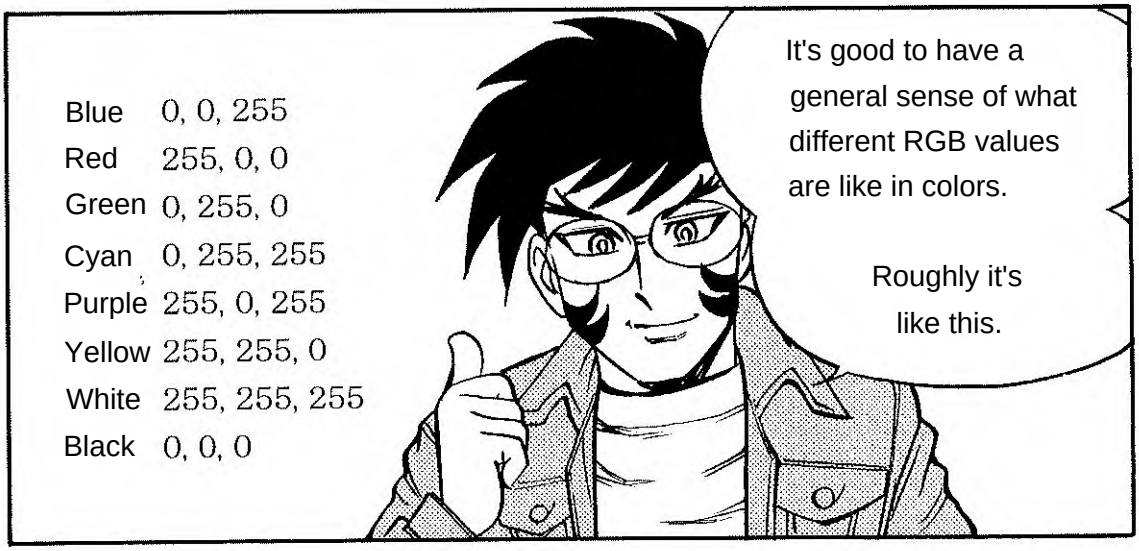

To change the background BGCOL is the command
Does it use the same 8 colors as TEXTCOL?
Next is to change the color of the background.
No, in this case, you use the three primary colors of light - red, green, blue, and specify their strength from 0 to 255.
Like this.
BGCOL 255,0,0
Woah, it turned totally red!
BGCOL 255,0,0

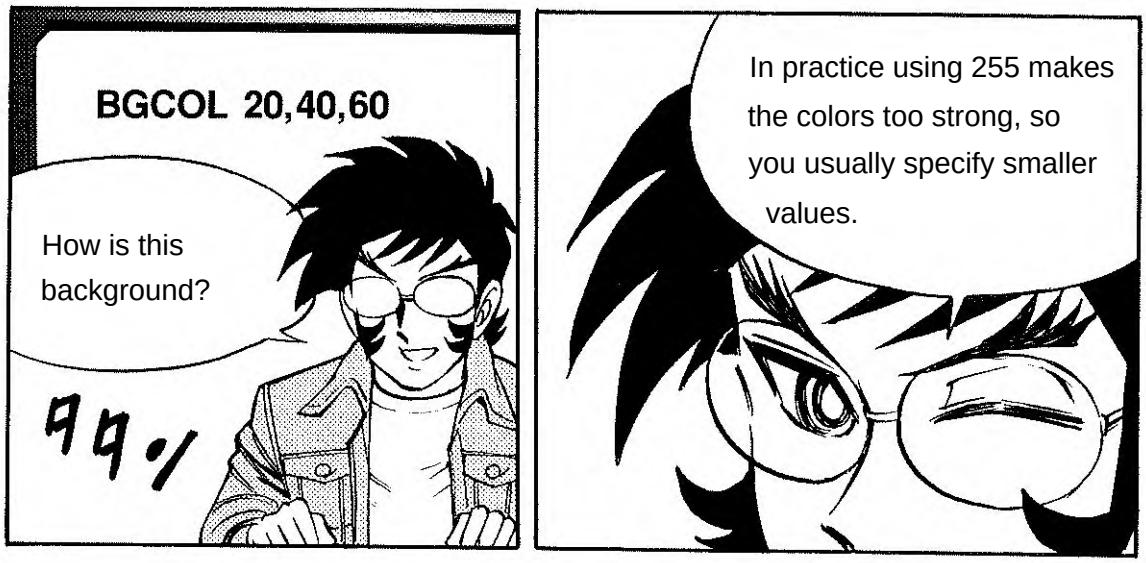
Oh, it returned to normal.
Red is at max strength.
BGCOL 255, 0, 0
Blue and green are set to 0.
Black would be this.
BGCOL 0,0,0
Since you can go from 0 to 255 with 256 steps for each of red green and blue, how many total colors can we make?
256X256X256 in total?
If I do that...
So there's three each with 256 options.

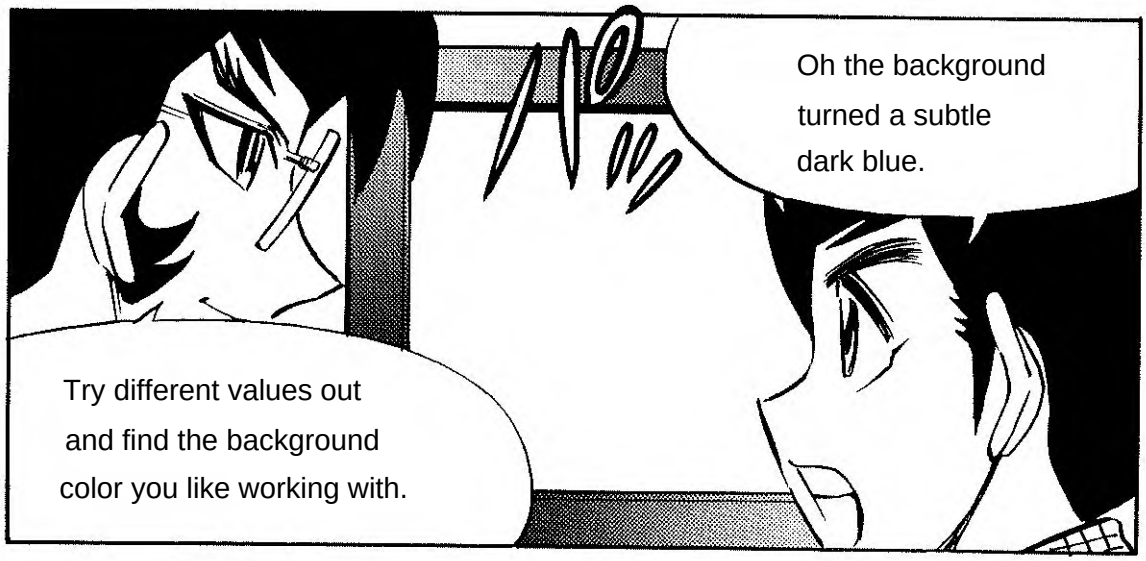

A total of 16777216 possible colors.
Woah, that's a huge number!

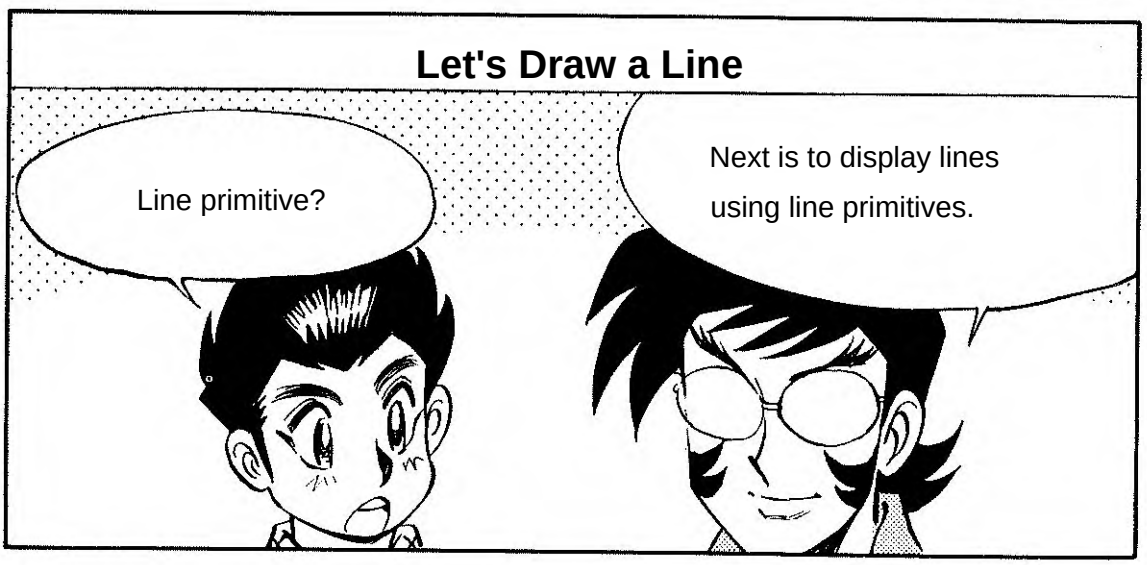

These three values are referred to as RGB
Its an acronym of RED • GREEN • BLUE
RED (赤) · GREEN (緑) · BLUE (青) ⇒ RGB

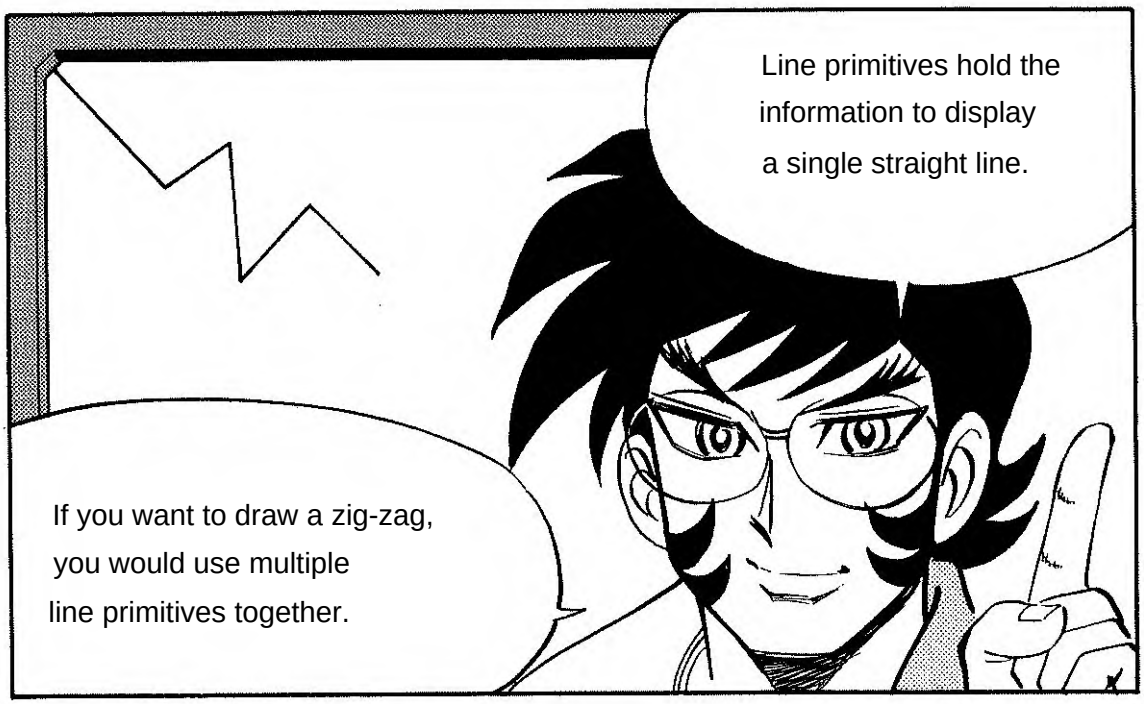
For graphics functions, you usally state colors with these three values.

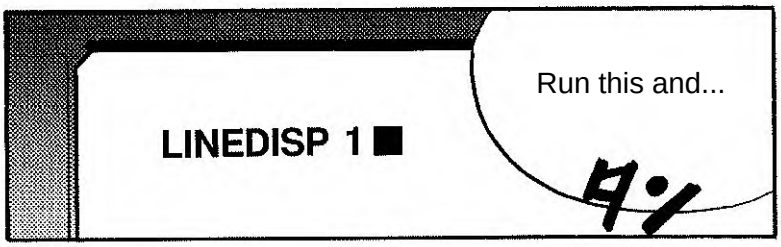

Blue 0, 0, 255
Red 255, 0, 0
Green 0, 255, 0
Cyan 0, 255, 255
Purple 255, 0, 255
Yellow 255, 255, 0
White 255, 255, 255
Black 0, 0, 0
It's good to have a general sense of what different RGB values are like in colors.
Roughly it's like this.


BGCOL 20,40,60
How is this background?
In practice using 255 makes the colors too strong, so you usually specify smaller values.

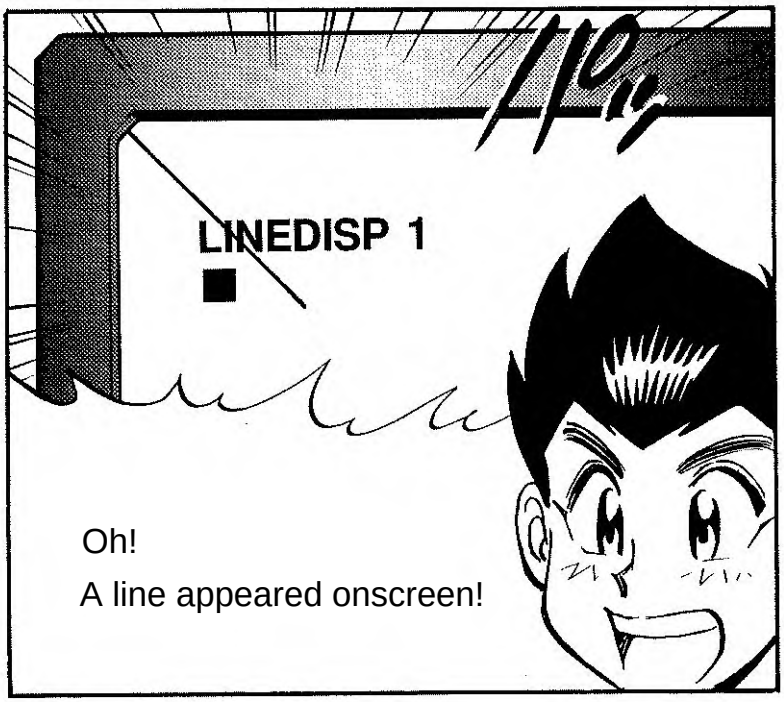

Oh the background turned a subtle dark blue.
Try different values out and find the background color you like working with.

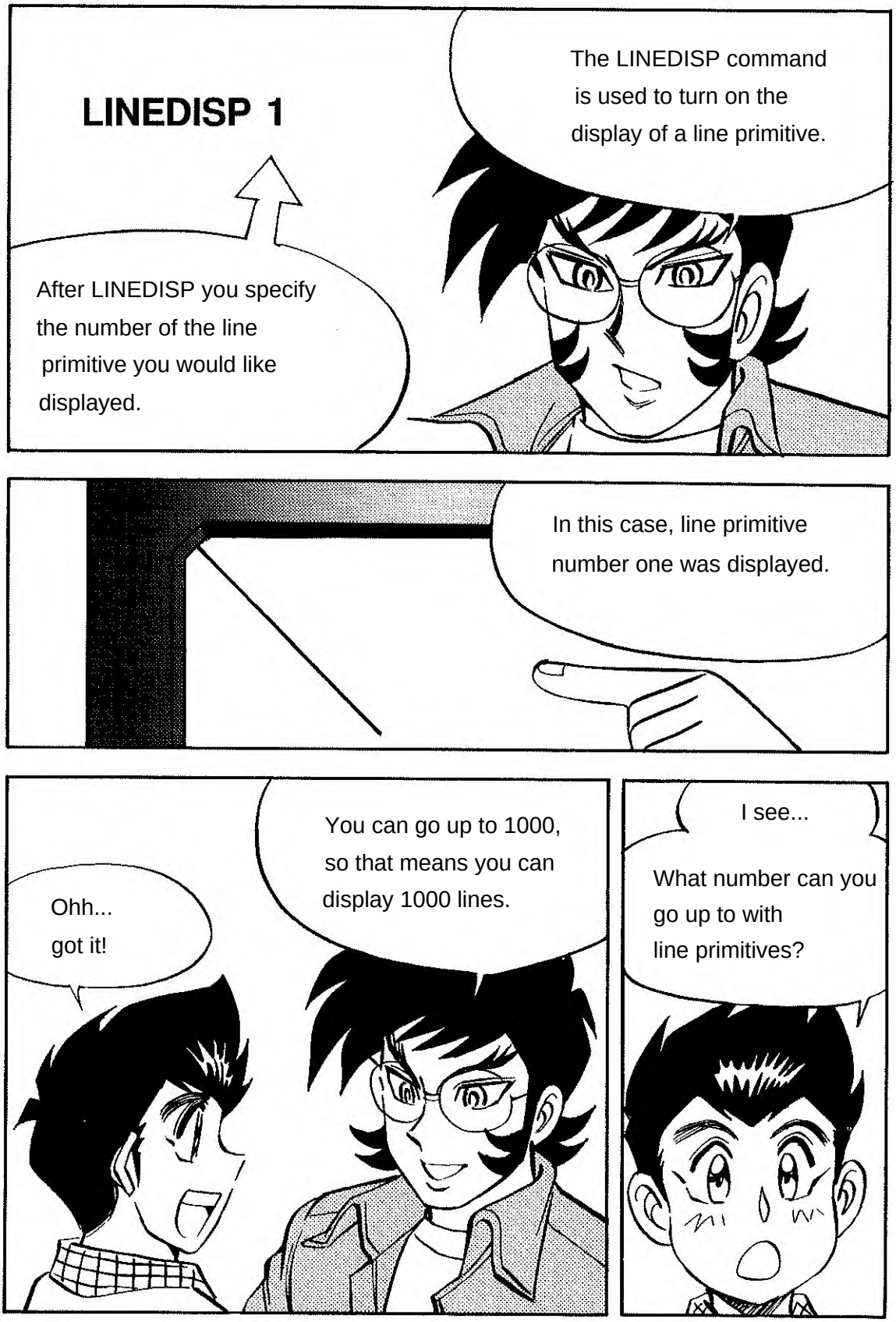

Let's Draw a Line
Line primitive?
Next is to display lines using line primitives.

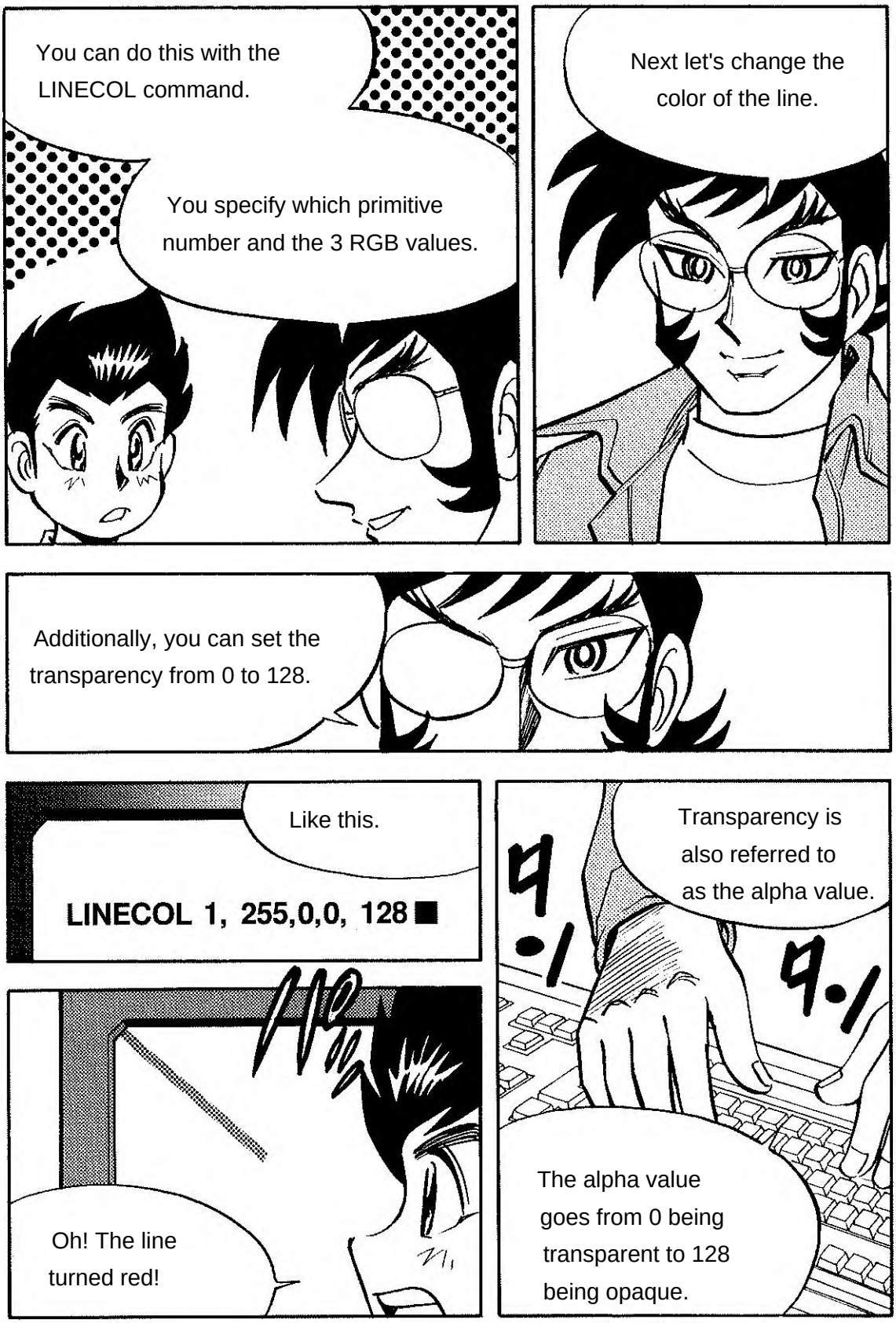

Line primitives hold the information to display a single straight line.
If you want to draw a zig-zag, you would use multiple line primitives together.

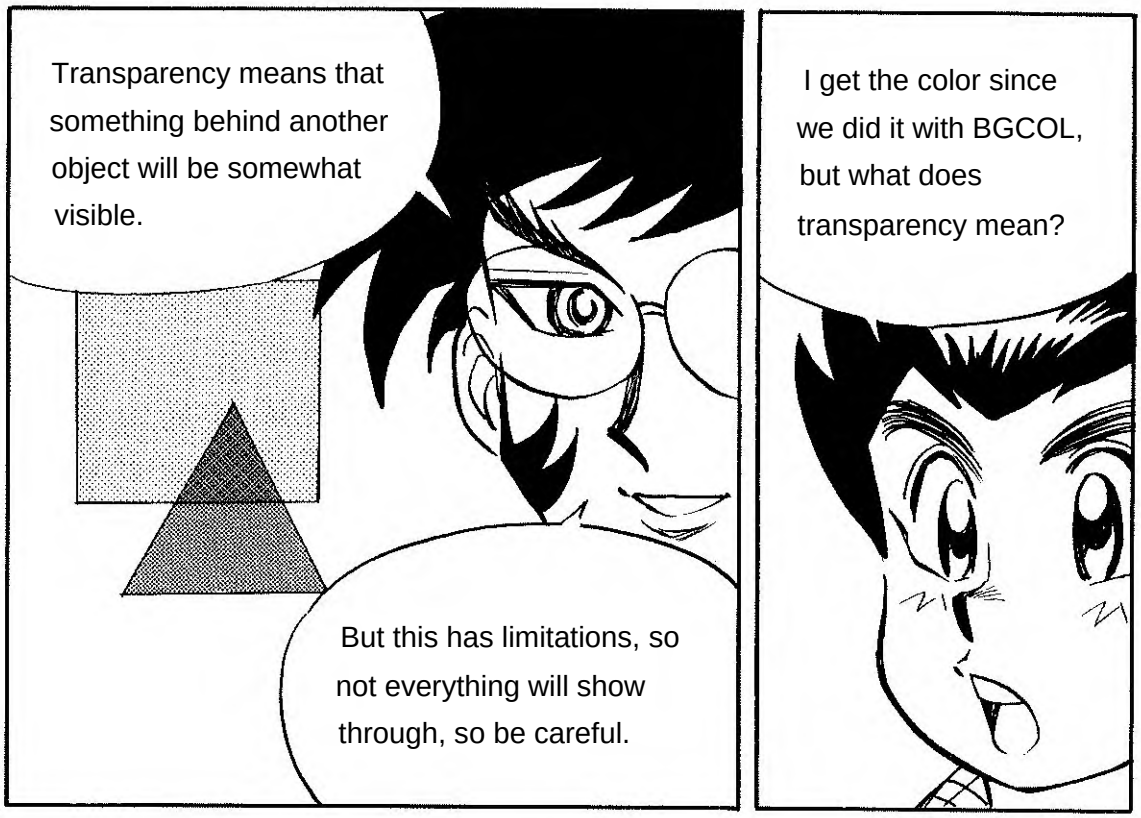

LINEDISP 1 ■
Run this and...

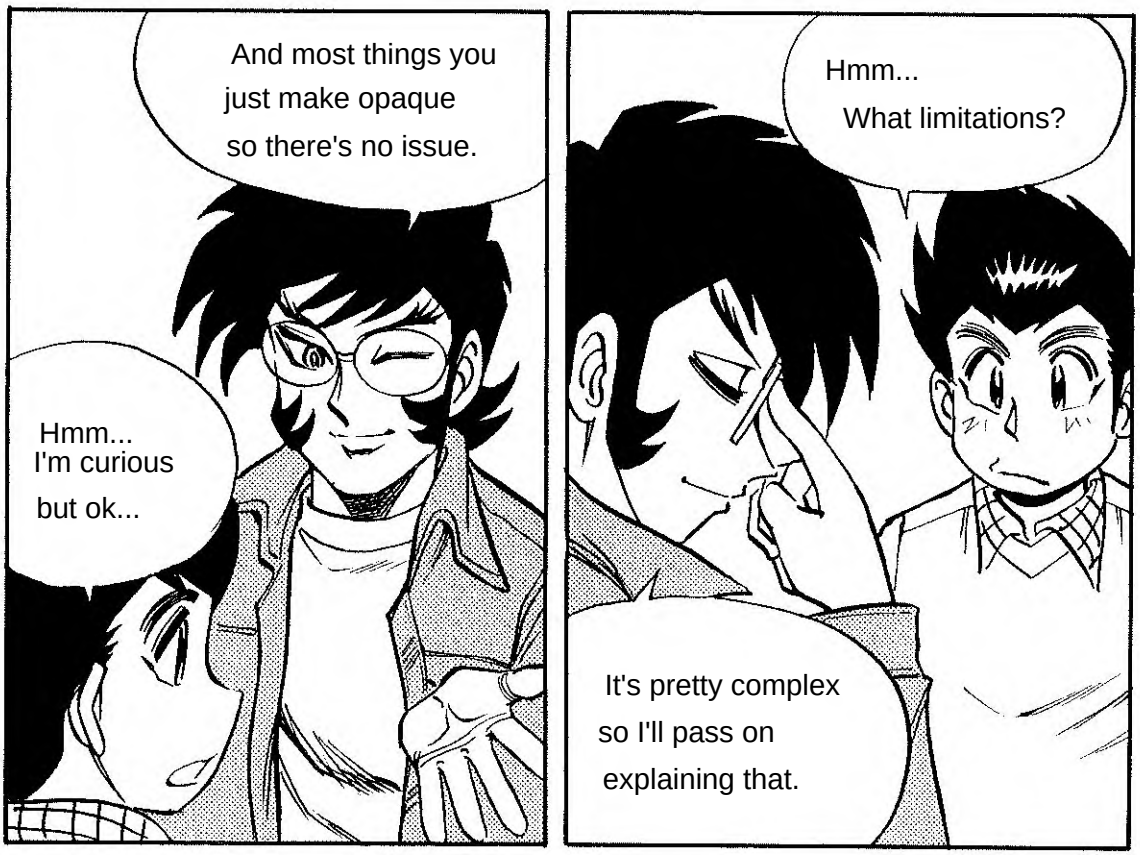
Oh...
Let's just give it a shot.


LINEDISP 1
■
Oh!
A line appeared onscreen!


LINEDISP 1
The LINEDISP command is used to turn on the display of a line primitive.
After LINEDISP you specify the number of the line primitive you would like displayed.
In this case, line primitive number one was displayed.
You can go up to 1000, so that means you can display 1000 lines.
Ohh... got it!
I see...
What number can you go up to with line primitives?

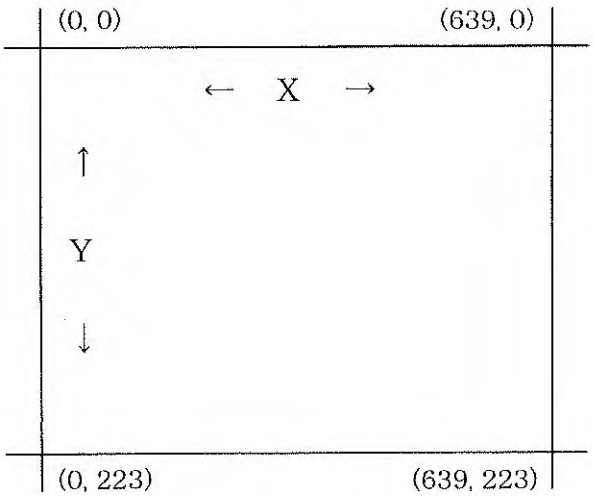

You can do this with the LINECOL command.
You specify which primitive number and the 3 RGB values.
Next let's change the color of the line.
Additionally, you can set the transparency from 0 to 128.
Like this.
LINECOL 1, 255,0,0, 128
Oh! The line turned red!
Transparency is also referred to as the alpha value.
The alpha value goes from 0 being transparent to 128 being opaque.


Transparency means that something behind another object will be somewhat visible.
But this has limitations, so not everything will show through, so be careful.
I get the color since we did it with BGCOL, but what does transparency mean?


And most things you just make opaque so there's no issue.
Hmm... I'm curious but ok...
Hmm... What limitations?
It's pretty complex so I'll pass on explaining that.

Next let's change the position of the displayed line.

To change the position of a line primitive, you use the LINEPOS command.

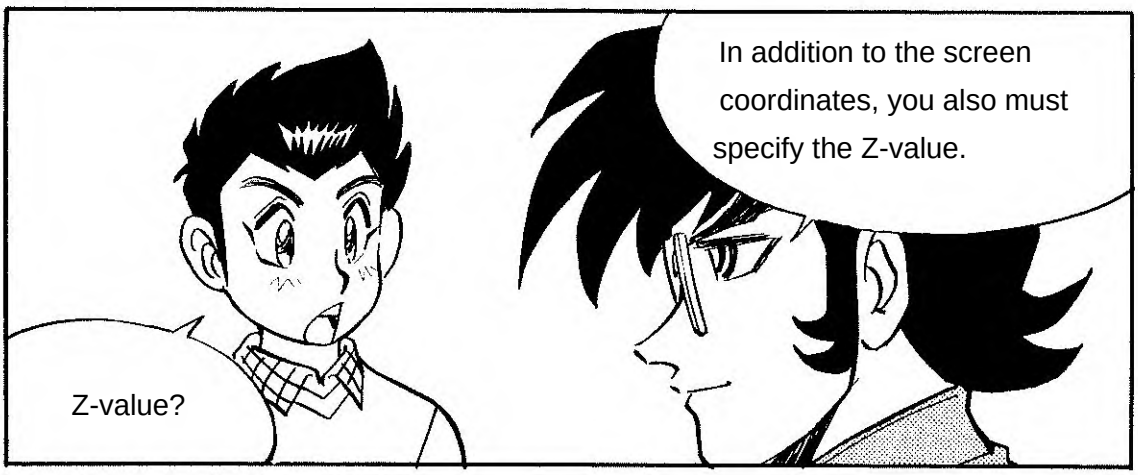


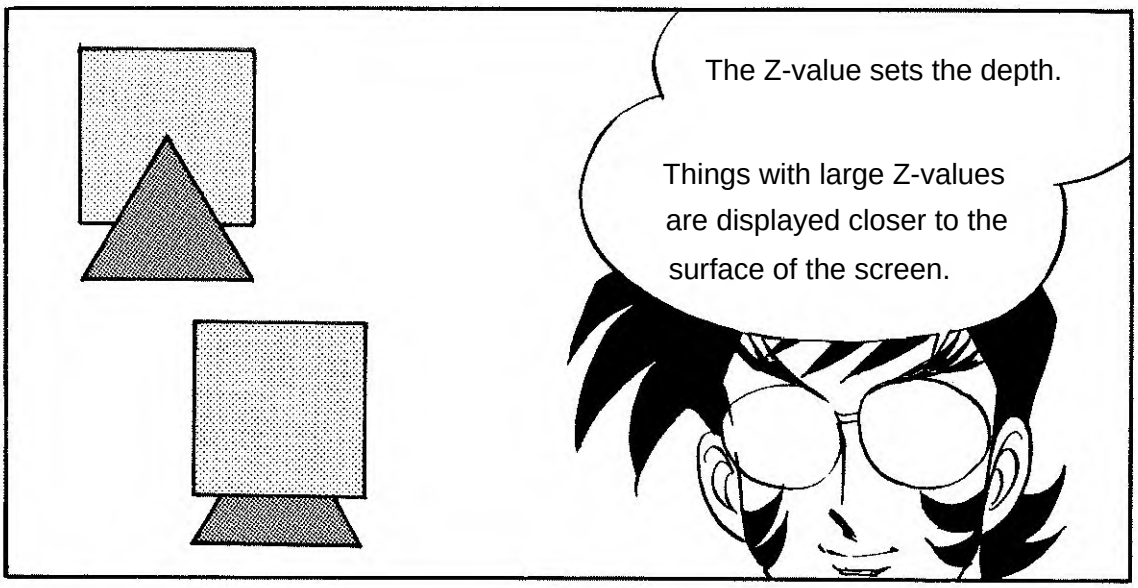
Screen coordinates are set with X and Y:
Top left (0, 0)
Top right (639, 0)
Bottom left (0, 223)
Bottom right (639, 223)

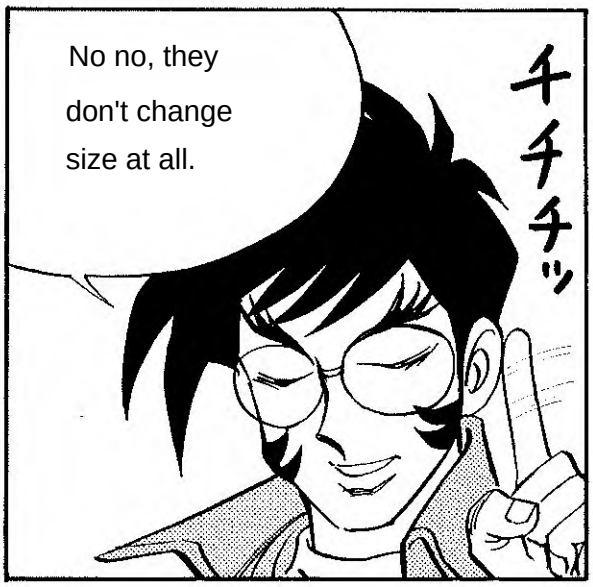
And if you look at the coordinates, it is divided more finely vertically than horizontally.

One coordinate is called one pixel, and each pixel is rectangular.

Depending on the TV the very edge might not be displayed so keep that in mind.

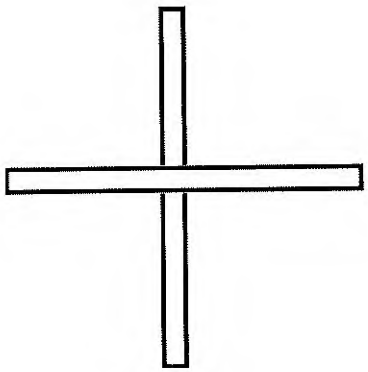

In addition to the screen coordinates, you also must specify the Z-value.
Z-value?


The Z-value sets the depth.
Things with large Z-values are displayed closer to the surface of the screen.

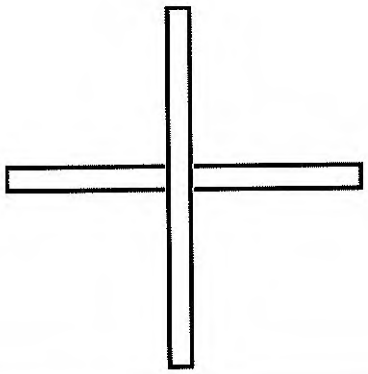

No no, they don't change size at all.
チチチッ

So do things with big Z-values get bigger as they get closer?

Horizontal line has larger Z-value



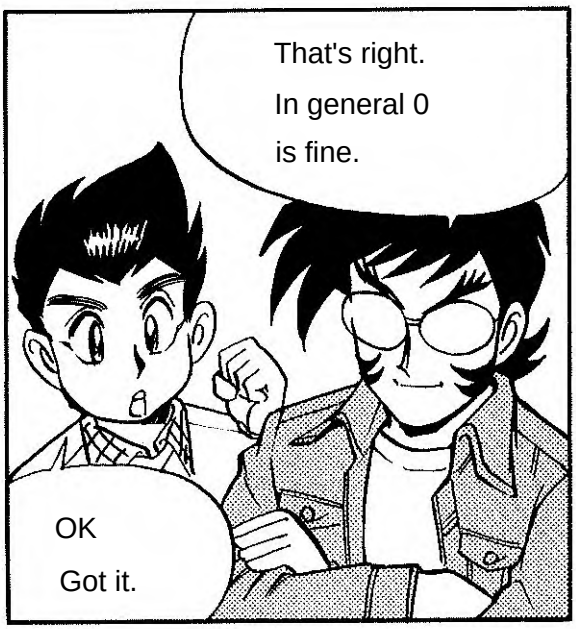
Vertical line has larger Z-value



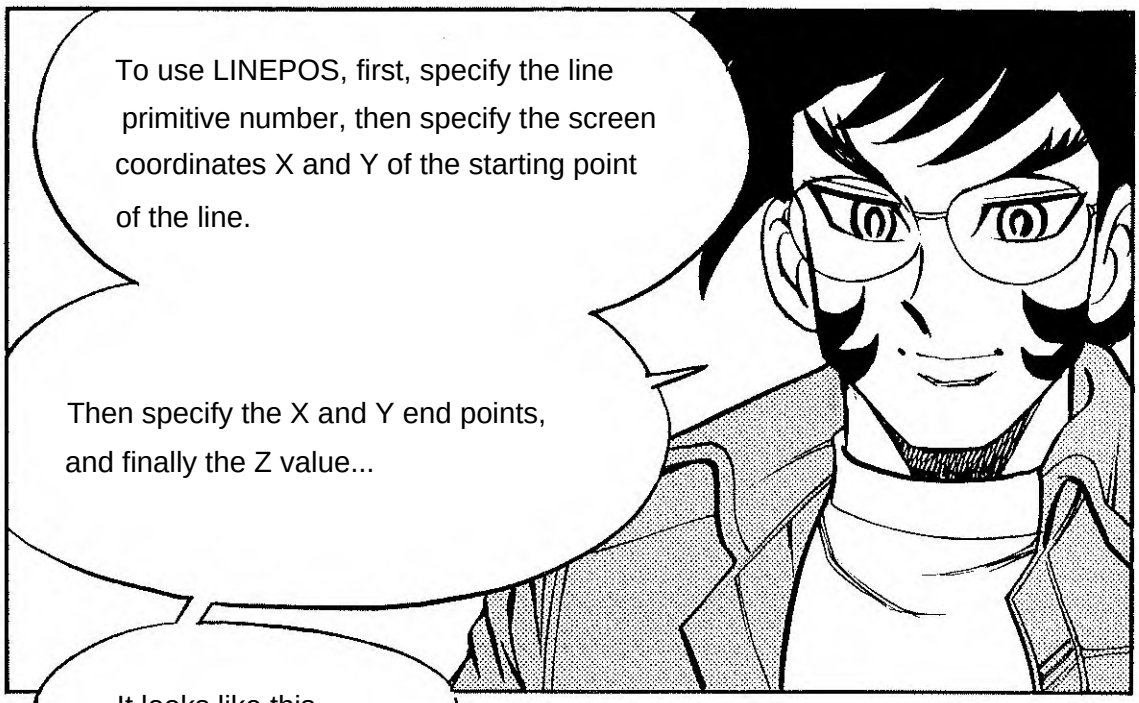
For example, two lines with different Z-values intersect. The one with the larger Z-value will be drawn on top of the other.

That's right. In general 0 is fine.

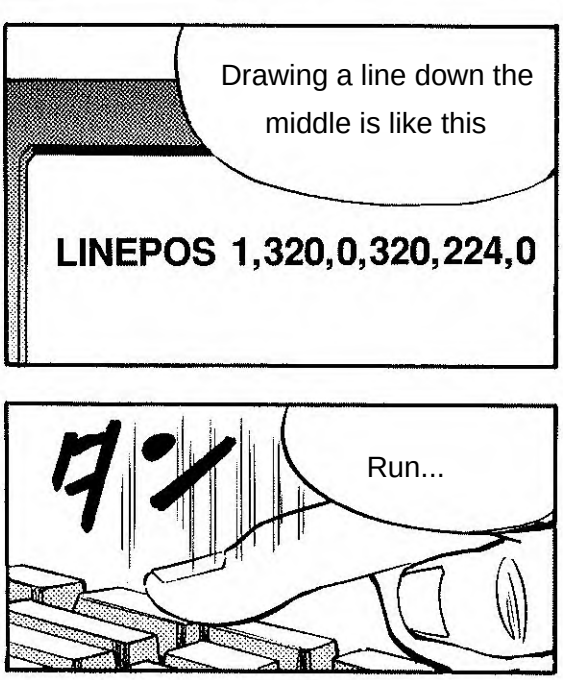
OK Got it.



I see, so it just sets which things get drawn on top of others.


To use LINEPOS, first, specify the line primitive number, then specify the screen coordinates X and Y of the starting point of the line.
Then specify the X and Y end points, and finally the Z value...
It looks like this.

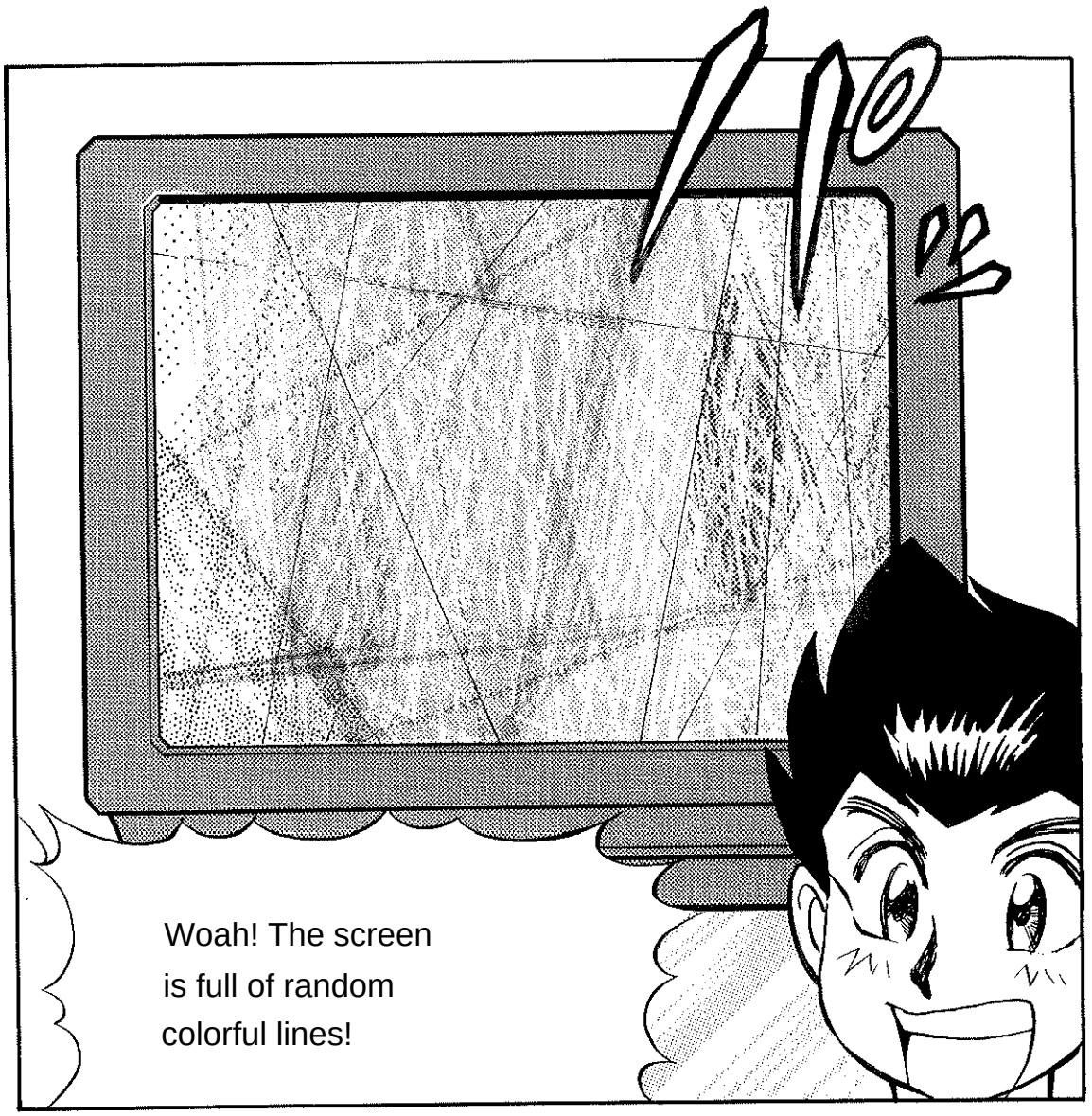
LINEPOS Line number, start X, start Y, end X, end Y, Z-value

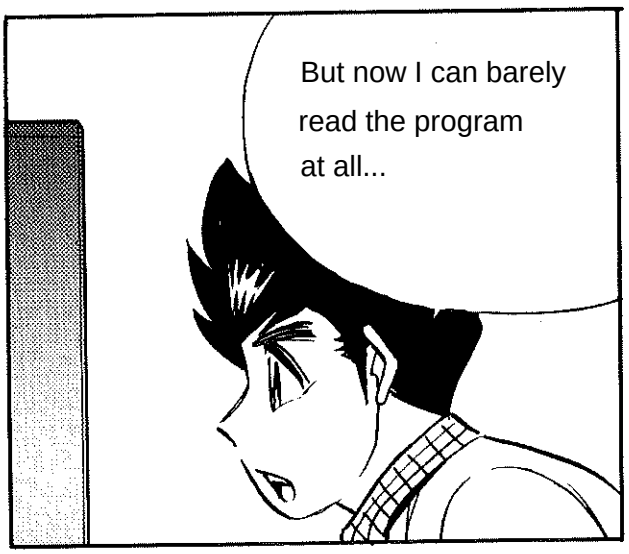


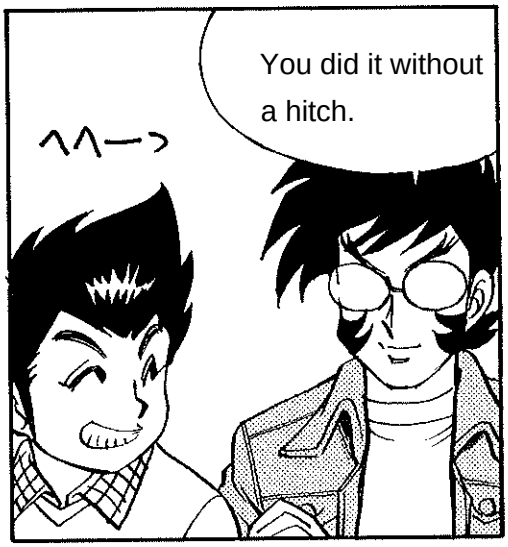
LINEOFF 1 ■
Execute...
If you want to erase a line primitive, use the LINEOFF command.
It's the opposite of LINEDISP.
LINEOFF 1
OK
■
I see, the line disappeared.
Here's a problem to solve.
Er, all of a sudden... Well I'll give it a try I guess...
I want you write a program to draw 1000 lines in random positions with random colors on screen.

* Try it yourself before looking at the answer.

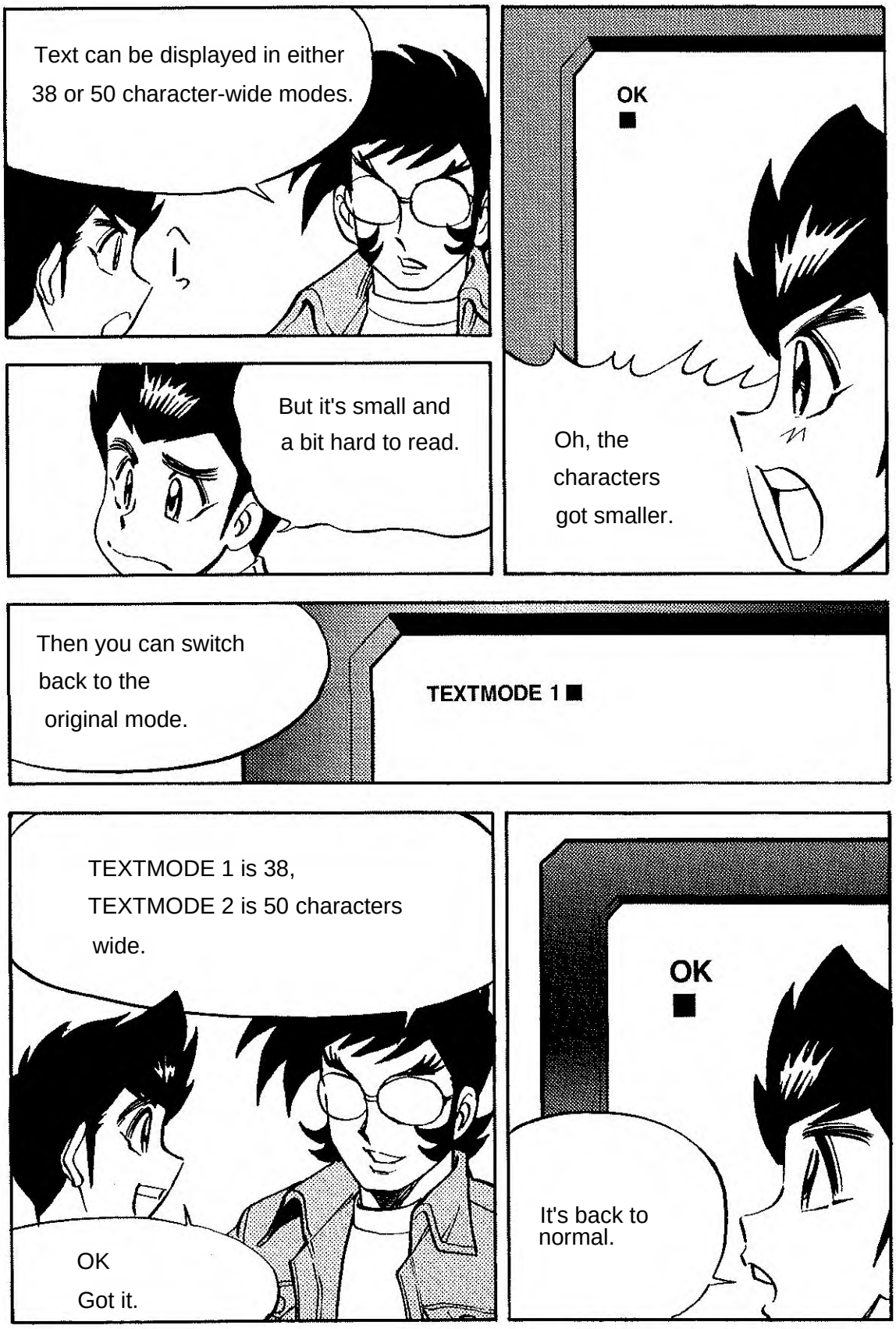
I'll try writing it out.
タン
RND
100
INT
FOR-
Well it's 1000 lines so I will use a FOR-NEXT loop with 1000 cycles.
The position and color are random so I'll use RND and multiply it, then round it off with INT.
100 FOR I=1 TO 1000
110 LINECOL I, INT(RND * 256), INT(RND * 256), INT(RND * 256), 128
120 LINEPOS I, INT(RND * 640), INT(RND * 224), INT(RND * 640), INT(RND * 224), 0
130 LINEDISP I
140 NEXT I
How's this?
RUN...
タン
Enter
Give it a go.


Woah! The screen is full of random colorful lines!


But now I can barely read the program at all...


You did it without a hitch.
ヘヘーっ

Oh, the screen returned to normal.
This is when INIT comes in.
INIT ■
INIT resets various things back to their original state, including things drawn on screen.
I see.
I'll teach you one more thing here.
TEXTMODE 2 ■
The TEXTMODE command.


Text can be displayed in either 38 or 50 character-wide modes.
OK
■
But it's small and a bit hard to read.
Oh, the characters got smaller.
Then you can switch back to the original mode.
TEXTMODE 1 ■
TEXTMODE 1 is 38, TEXTMODE 2 is 50 characters wide.
OK
Got it.
OK
■
It's back to normal.

LINEDISP ⇒ TILEDISP
LINEOFF ⇒ TILEOFF
LINECOL ⇒ TILECOL
LINEPOS ⇒ TILEPOS

They work the same way as line primitives.
All you have to do is change LINE to TILE.

Next I'll teach you how to draw rectangles.

This is done with tile primitives.

The specified arguments are the same as well, and there are again 1000 to work with.

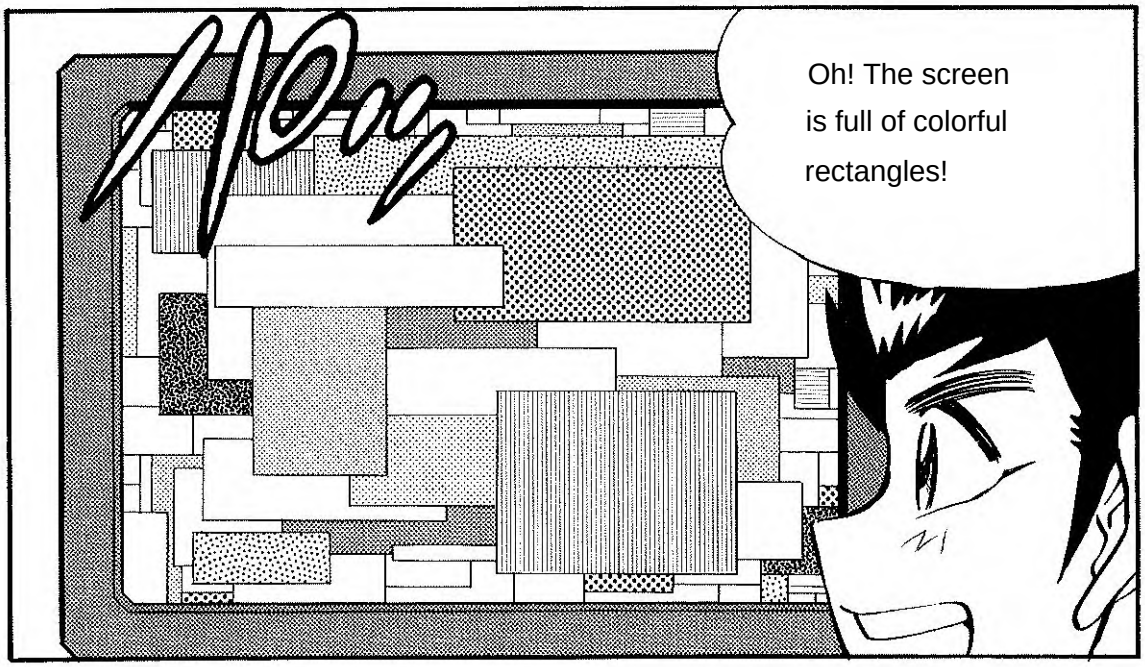
Let's rewrite the last program to use tiles instead.

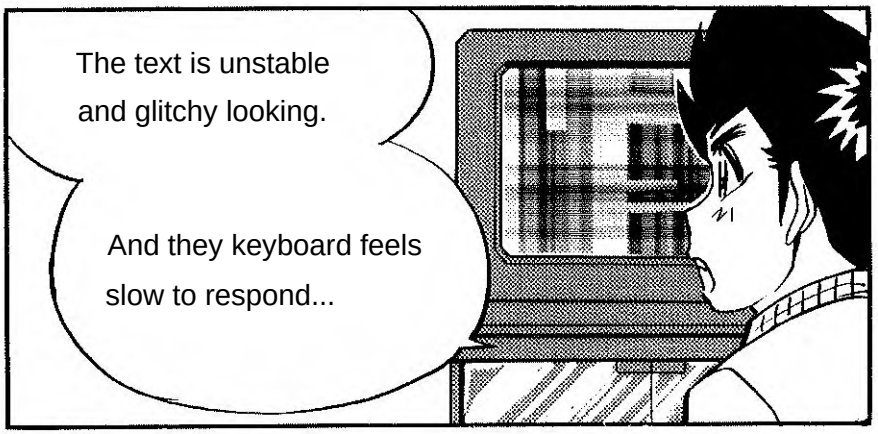
カチャカチャ

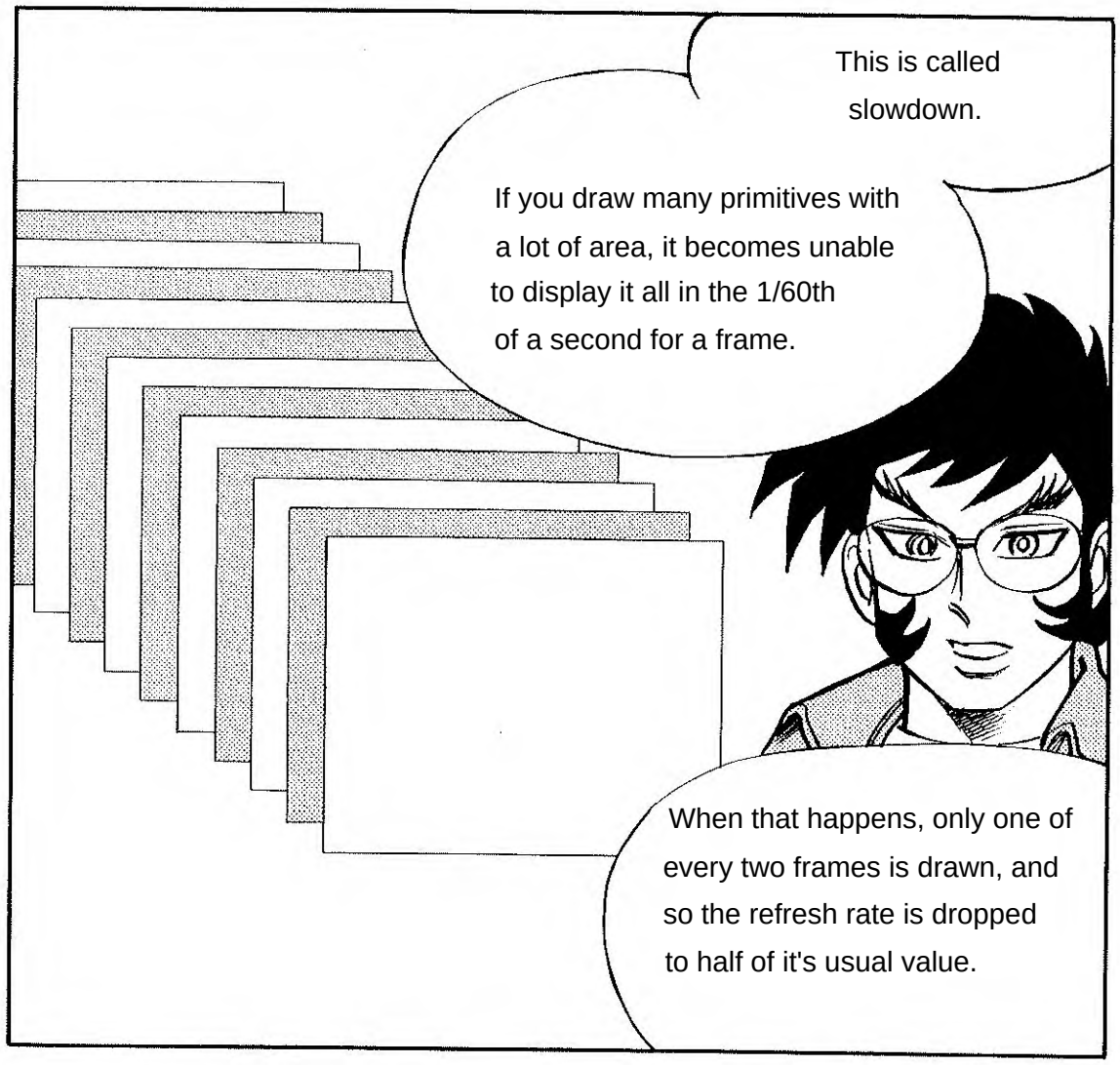
The two coordinates set the diagonal of the rectangle to be drawn, like this.

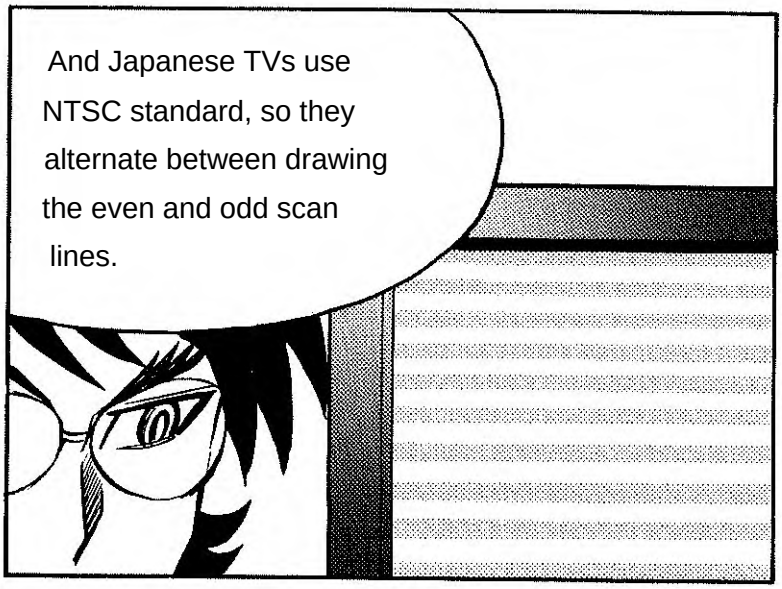
Execute with RUN.

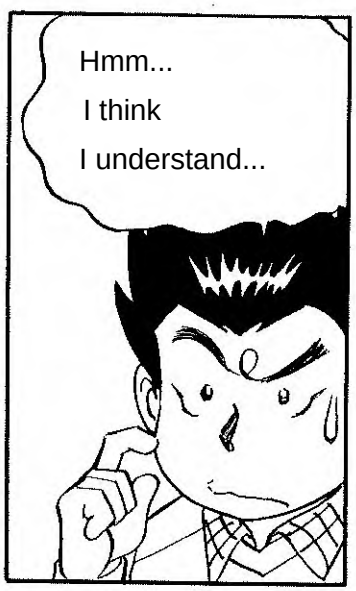
```
100 FOR I=1 TO 1000
110   TILECOL I, INT(RND * 256), INT(RND * 256),
INT(RND * 256), 128
120   TILEPOS I, INT(RND * 640), INT(RND * 224),
INT(RND * 640), INT(RND * 224), 0
130   TILEDISP I
140 NEXT I
■
```

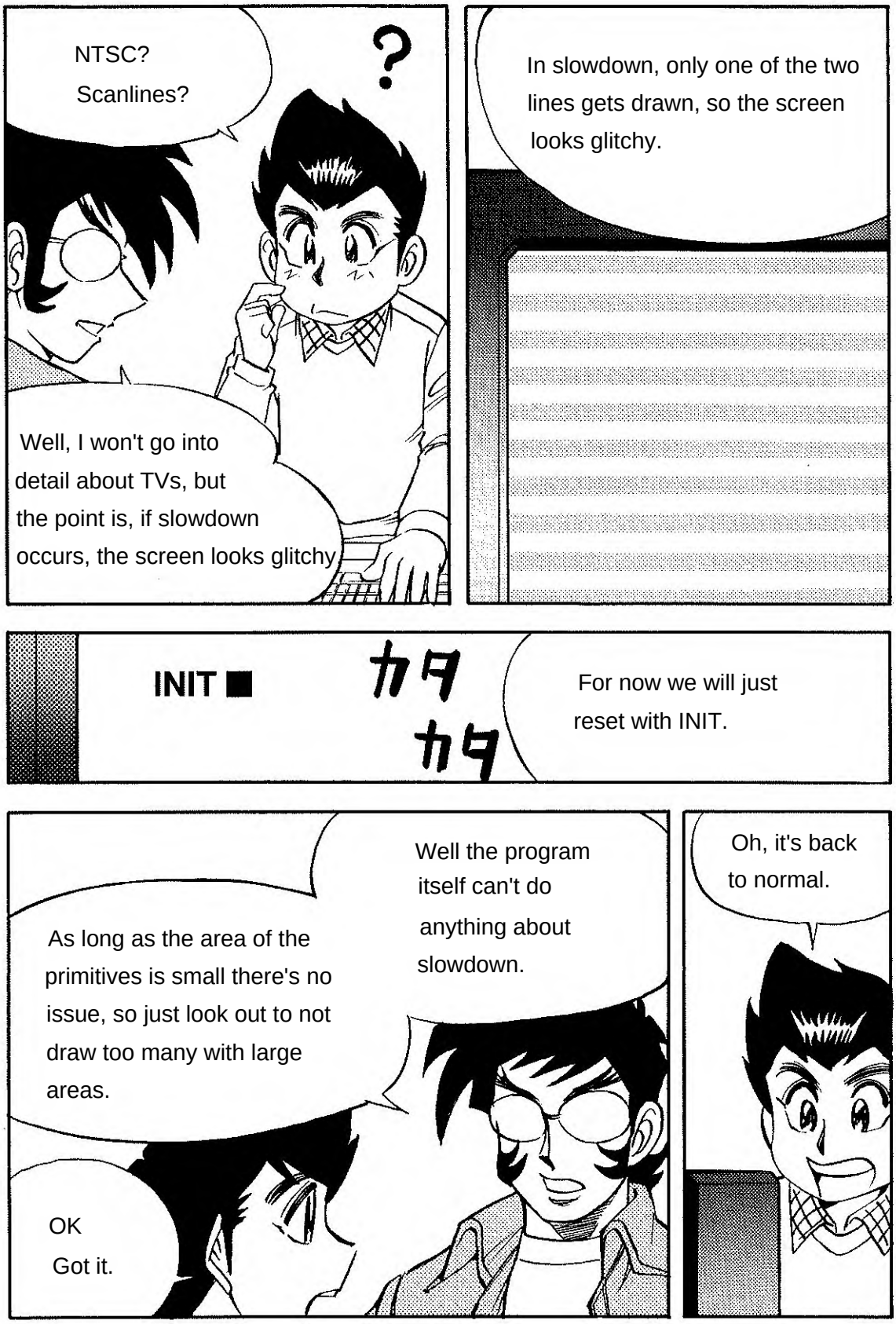





This is called slowdown.
If you draw many primitives with a lot of area, it becomes unable to display it all in the 1/60th of a second for a frame.
When that happens, only one of every two frames is drawn, and so the refresh rate is dropped to half of it's usual value.


And Japanese TVs use NTSC standard, so they alternate between drawing the even and odd scan lines.


Hmm... I think I understand...


NTSC?
Scanlines?
?
Well, I won't go into detail about TVs, but the point is, if slowdown occurs, the screen looks glitchy
In slowdown, only one of the two lines gets drawn, so the screen looks glitchy.
INIT ■
カタ
カタ
For now we will just reset with INIT.
As long as the area of the primitives is small there's no issue, so just look out to not draw too many with large areas.
Well the program itself can't do anything about slowdown.
OK
Got it.
Oh, it's back to normal.

Sprite Primitives

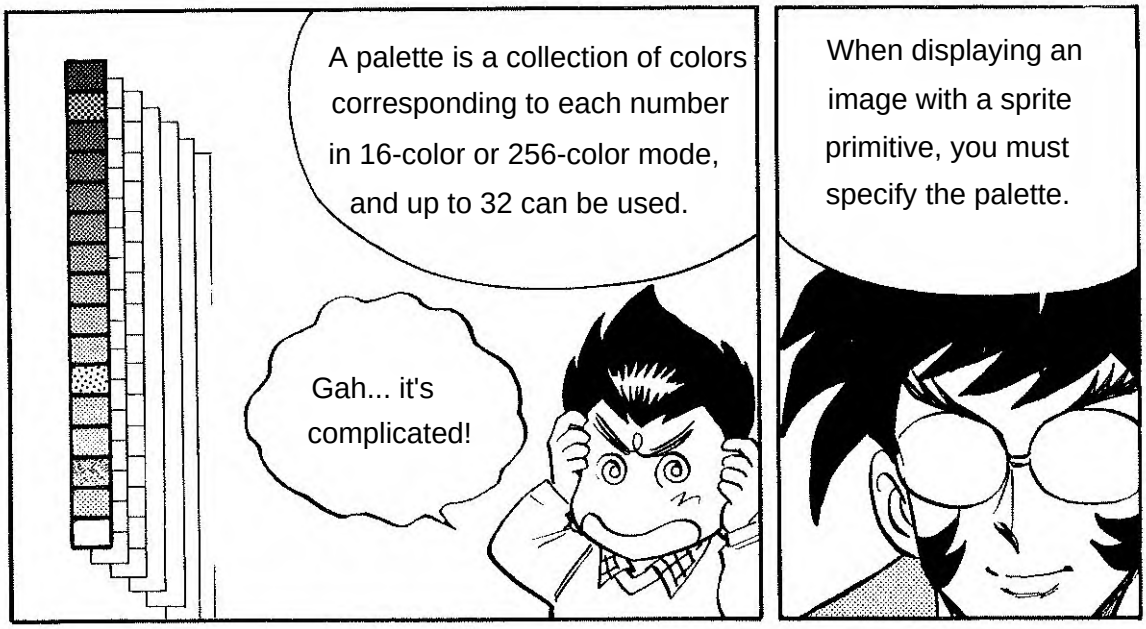
Next are sprite primitives. They are a bit more complicated than what we have done so far.

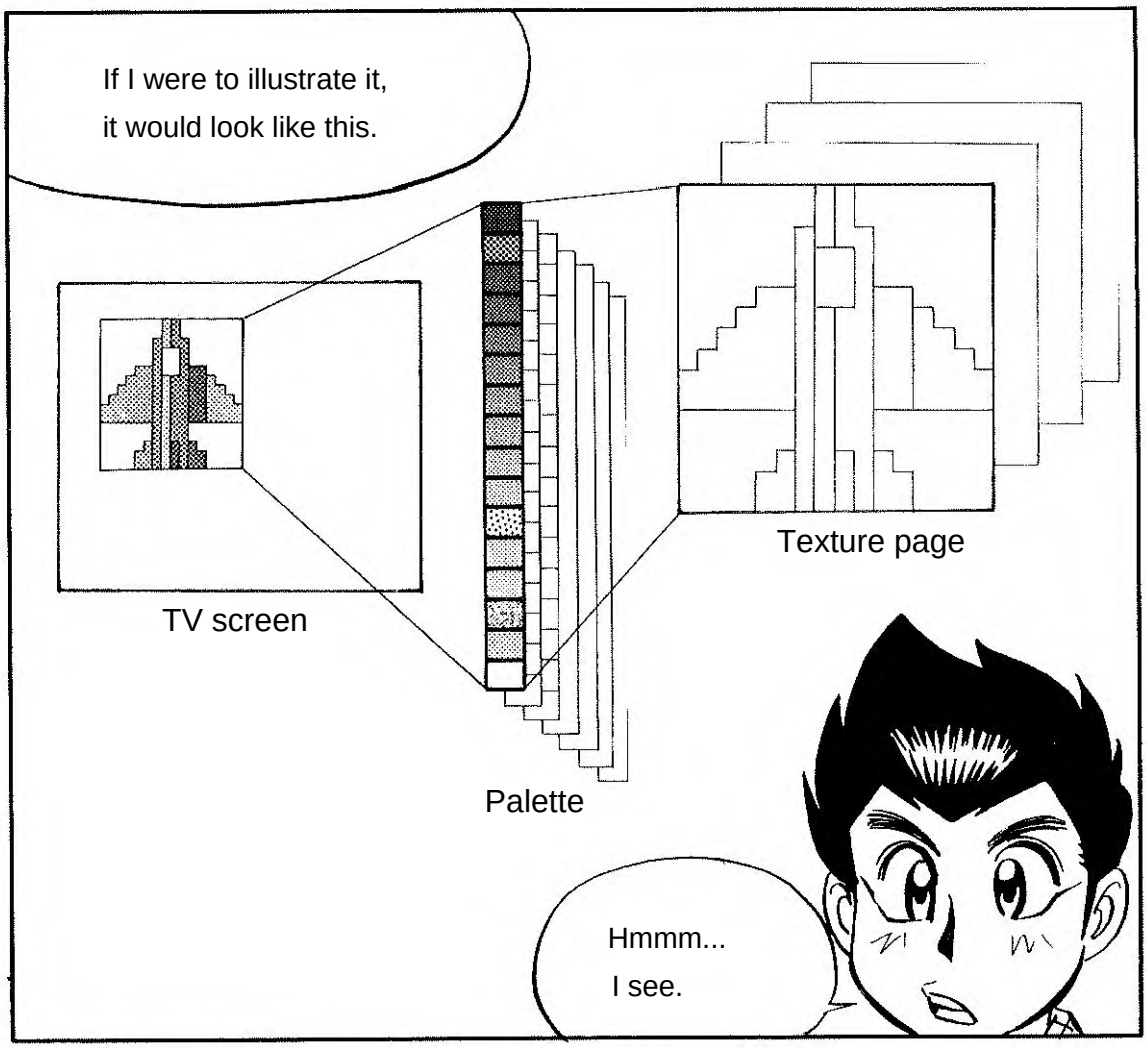
Let's leave tile primitives at that.

Sprite primitives display pictures from a texture page.

A texture page is where image data is saved, and there are 16 pages available.

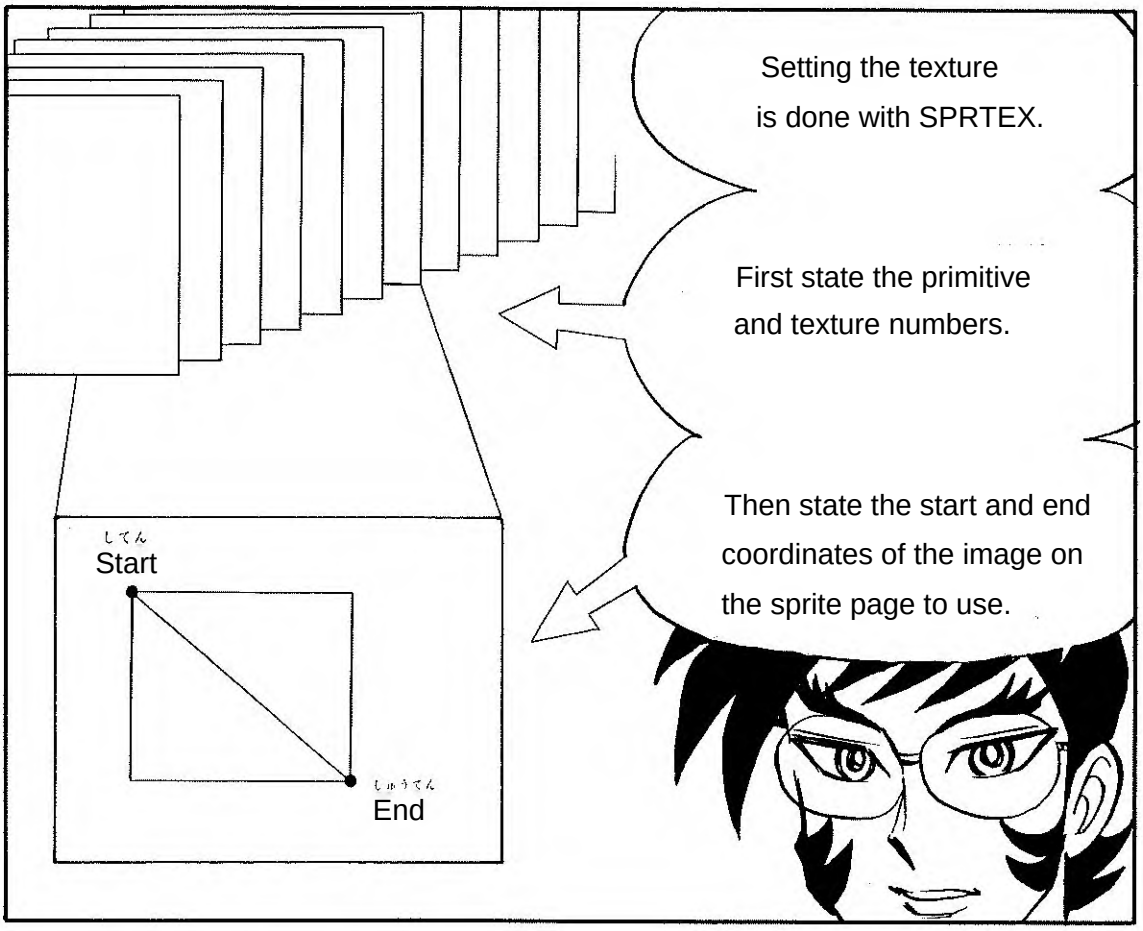
Images on texture pages are either 16 or 256 colors, and one page is 512x256 in size in 16 color mode, or 256x256 in 256 color mode.


A palette is a collection of colors corresponding to each number in 16-color or 256-color mode, and up to 32 can be used.
Gah... it's complicated!
When displaying an image with a sprite primitive, you must specify the palette.

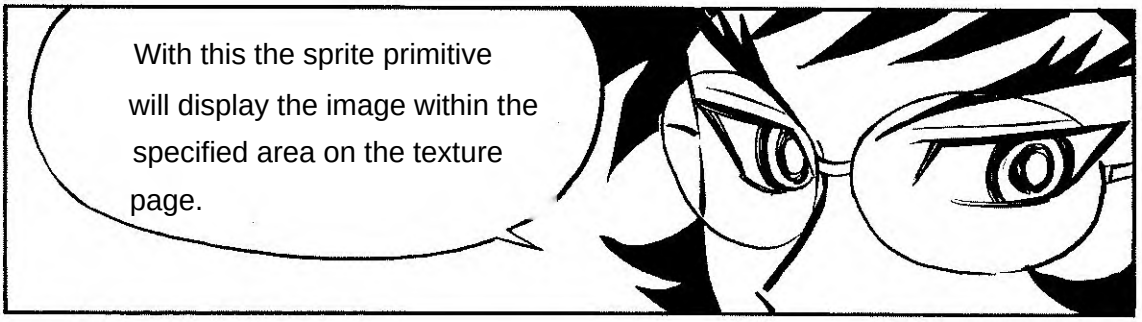

If I were to illustrate it, it would look like this.
TV screen
Palette
Texture page
Hmmm... I see.

There are pictures in texture pages and palettes preset so you don't have to make your own right away, though.
Read about the sample materials more in the manual.
To make texture pages and palettes, use IMAGE-SKETCH.
If you load data you made yourself, you can freely make your own images and palettes.
Got it.
To load, use the TEXLOAD and PALLOAD commands.
I won't explain here though.

Different from other primitives, instead of specifying a color, you specify a texture and a palette.
Let's display an image using a sprite primitive.
Similar to other primitives, display with SPRDISP and hide with SPROFF.
SPRDISP 1■
Oh, something is being displayed.
Run...



SPRTEX Sprite number, Texture page number,
Texture X-start point, Texture Y-start point,
Texture X-end point, Texture Y-end point



Next to specify the palette.

SPRPAL Primitive number, Palette number, Offset-value, Alpha-mode

If the palette chosen doesn't match with the image on the texture page, the sprite can display with strange colors.

If the offset is zero, the first 16 of the 256 colors are used.

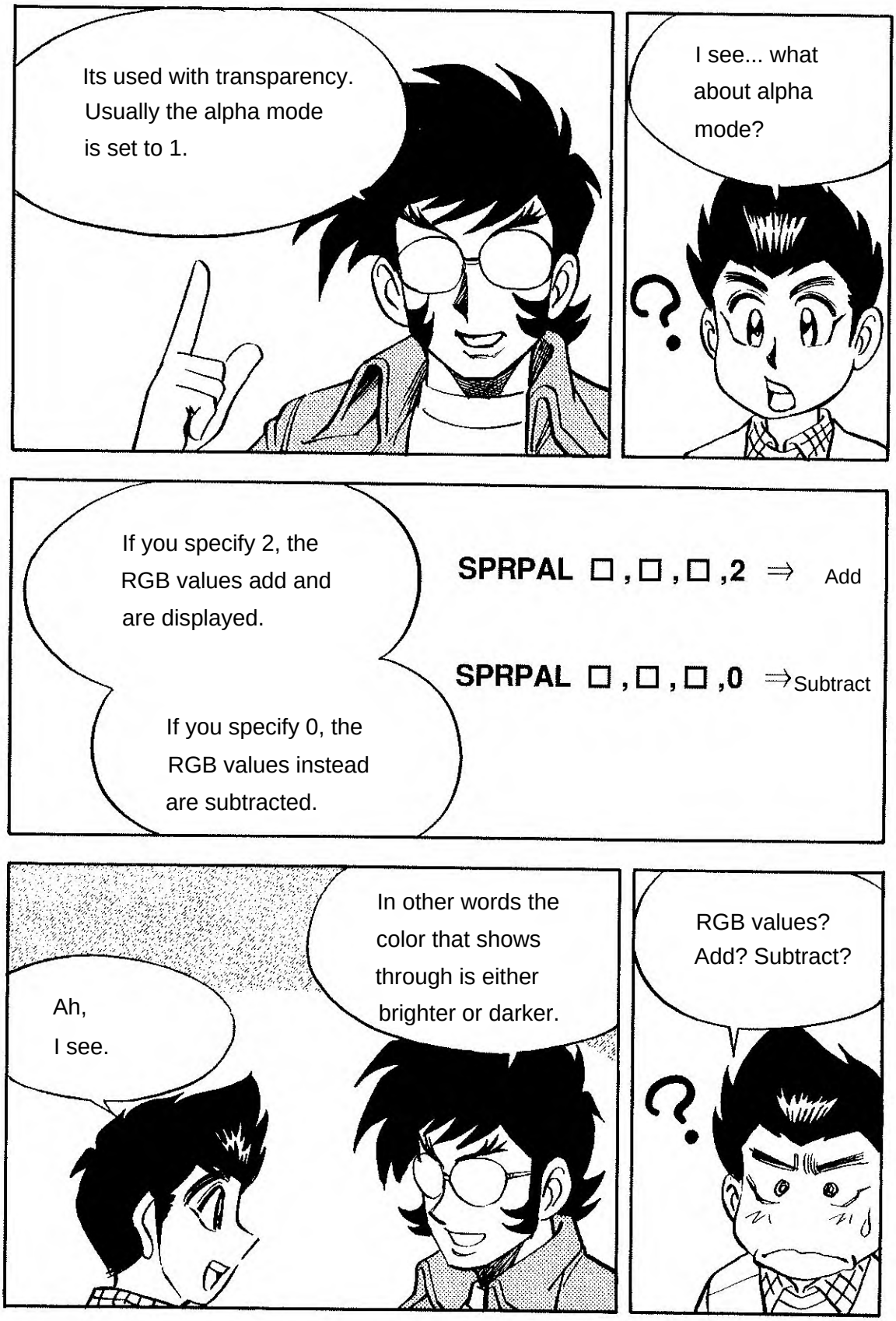
16 Colors

16 Colors

By adding one to the offset, you can shift the palette by 16 colors.

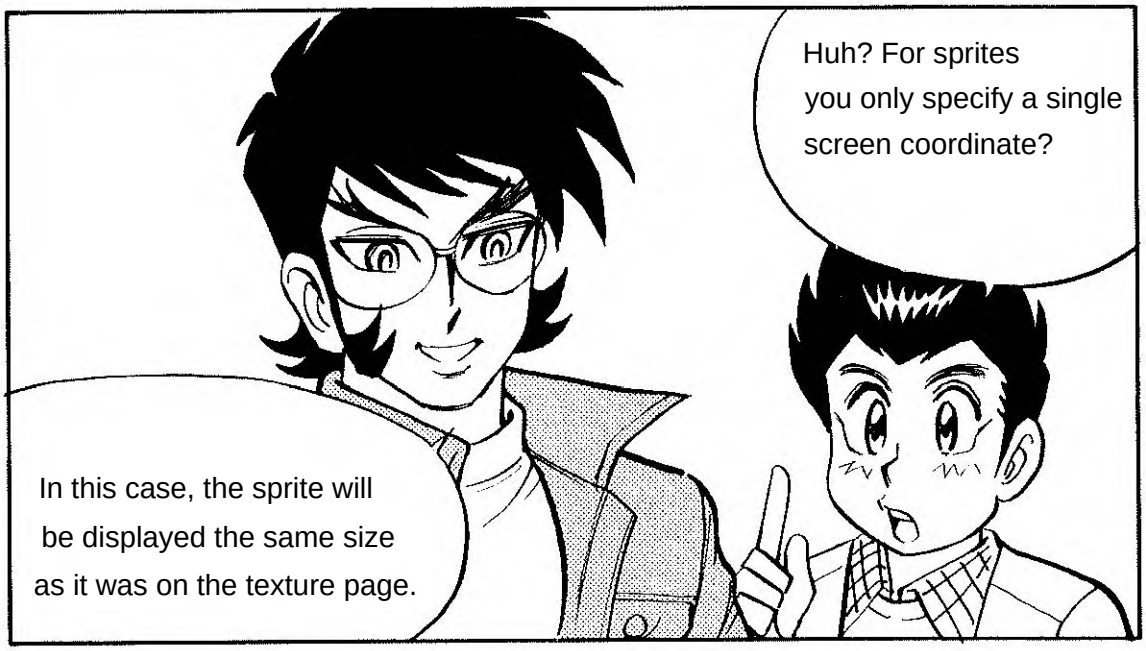
In one palette there are 256 colors, and so a 16 color texture makes use of an offset value to select the correct section.

Offset value?


Its used with transparency. Usually the alpha mode is set to 1.
I see... what about alpha mode?
If you specify 2, the RGB values add and are displayed.
SPRPAL □,□,□,2 ⇒ Add
SPRPAL □,□,□,0 ⇒Subtract
If you specify 0, the RGB values instead are subtracted.
Ah, I see.
In other words the color that shows through is either brighter or darker.
RGB values? Add? Subtract?

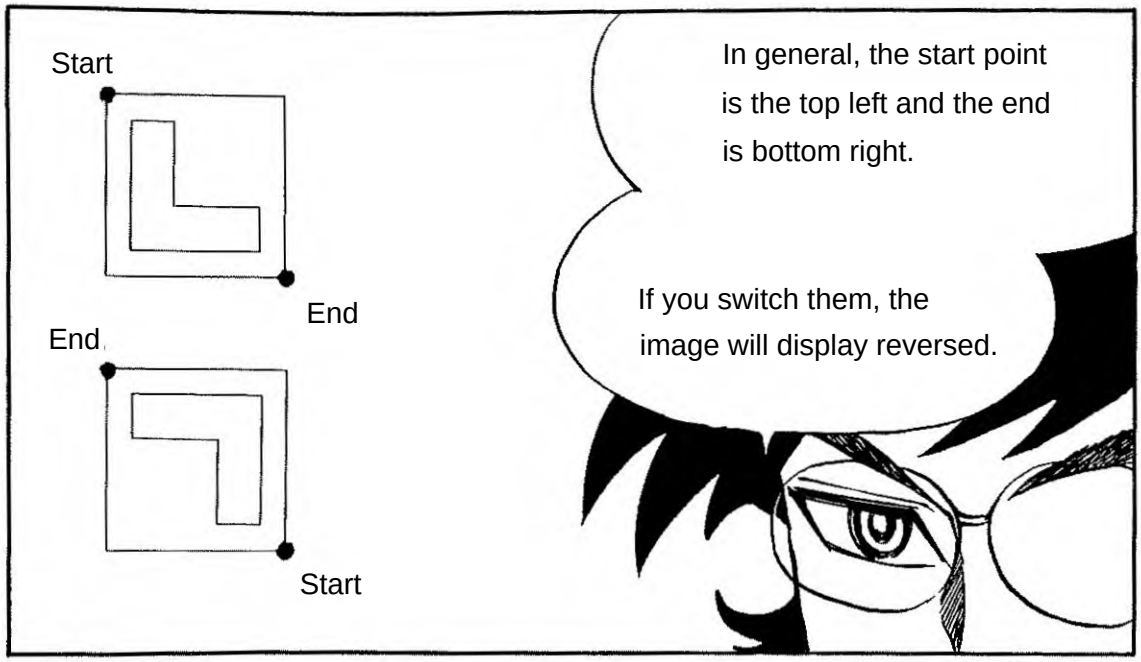
Next to set the position on screen, you use SPRPOS.

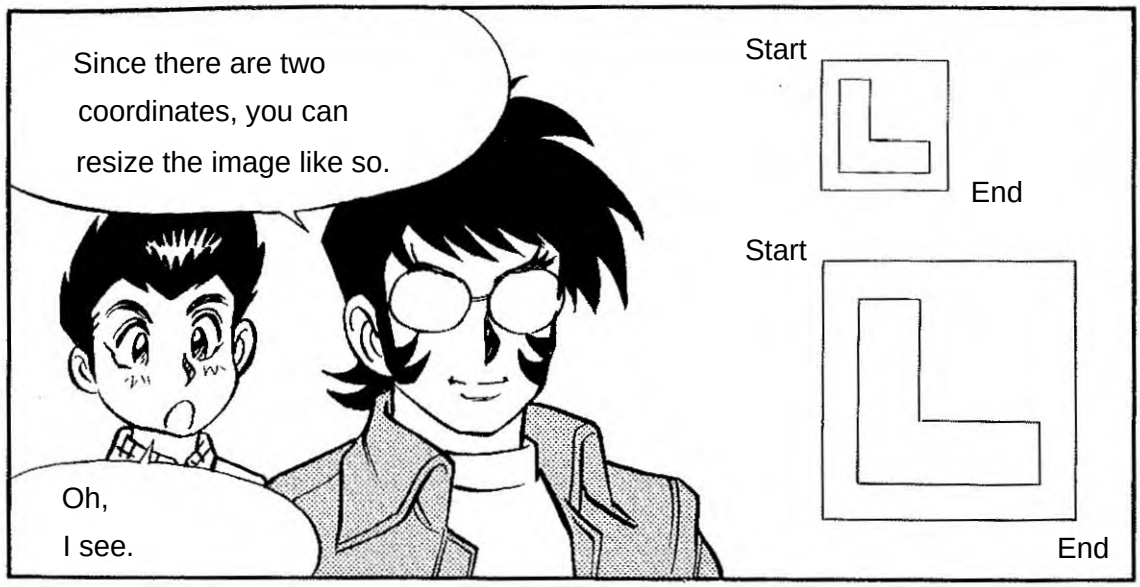
SPRPOS Primitive Number, Screen X, Screen Y, Z-value


Huh? For sprites you only specify a single screen coordinate?
In this case, the sprite will be displayed the same size as it was on the texture page.

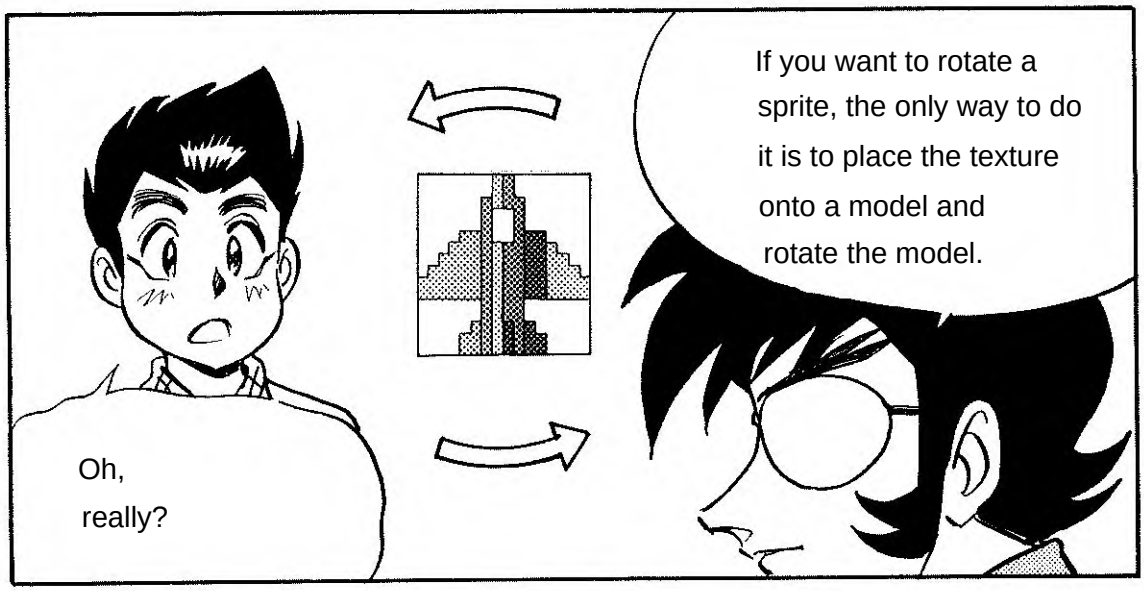
To shrink or enlarge it, use SPRPOS2. This command takes two coordinates.

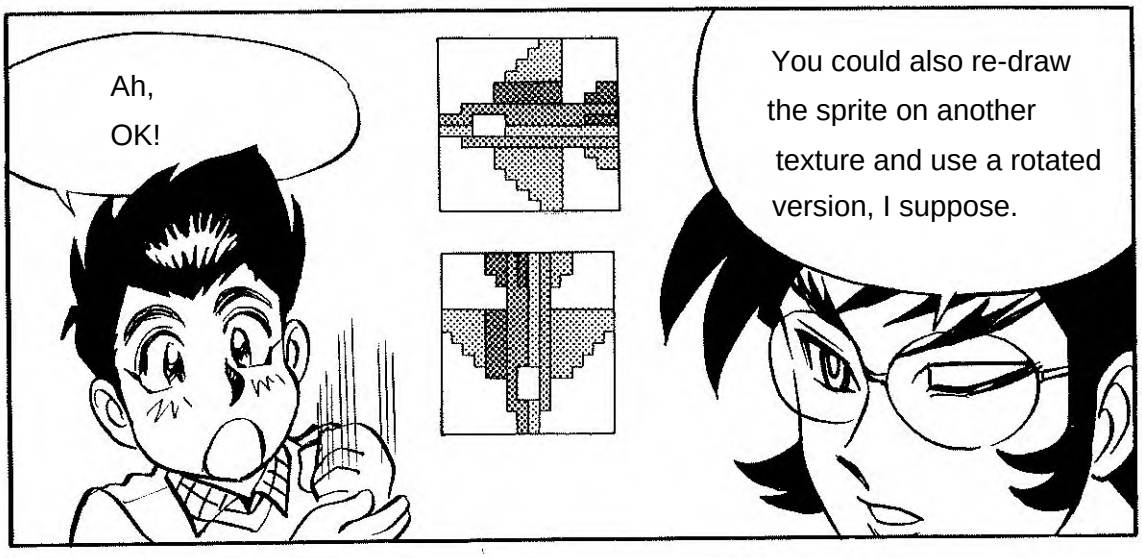
SPRPOS2 Primitive number, Start X, Start Y, End X, End Y, Z-value

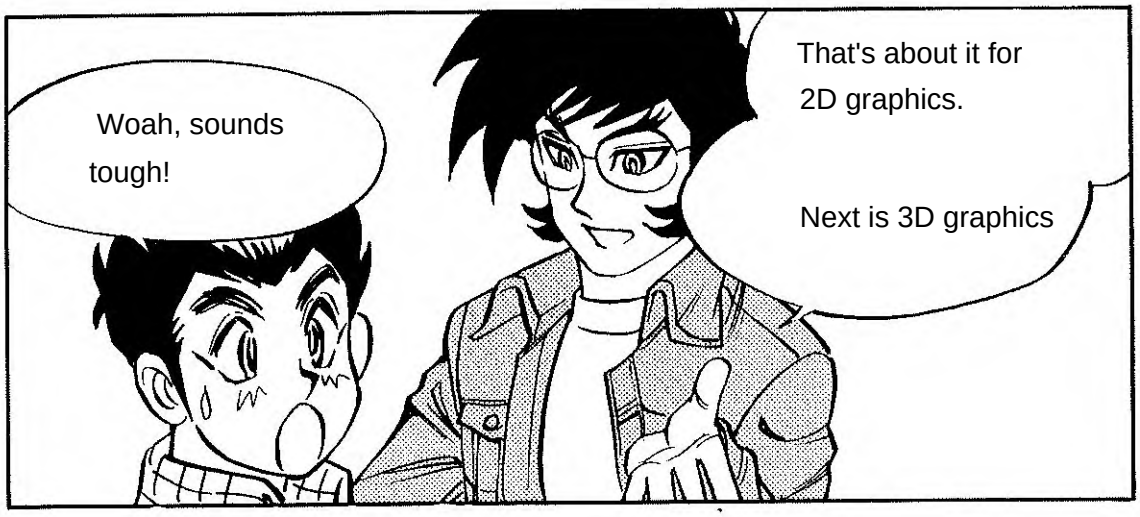

Start
End
End
Start
In general, the start point is the top left and the end is bottom right.
If you switch them, the image will display reversed.

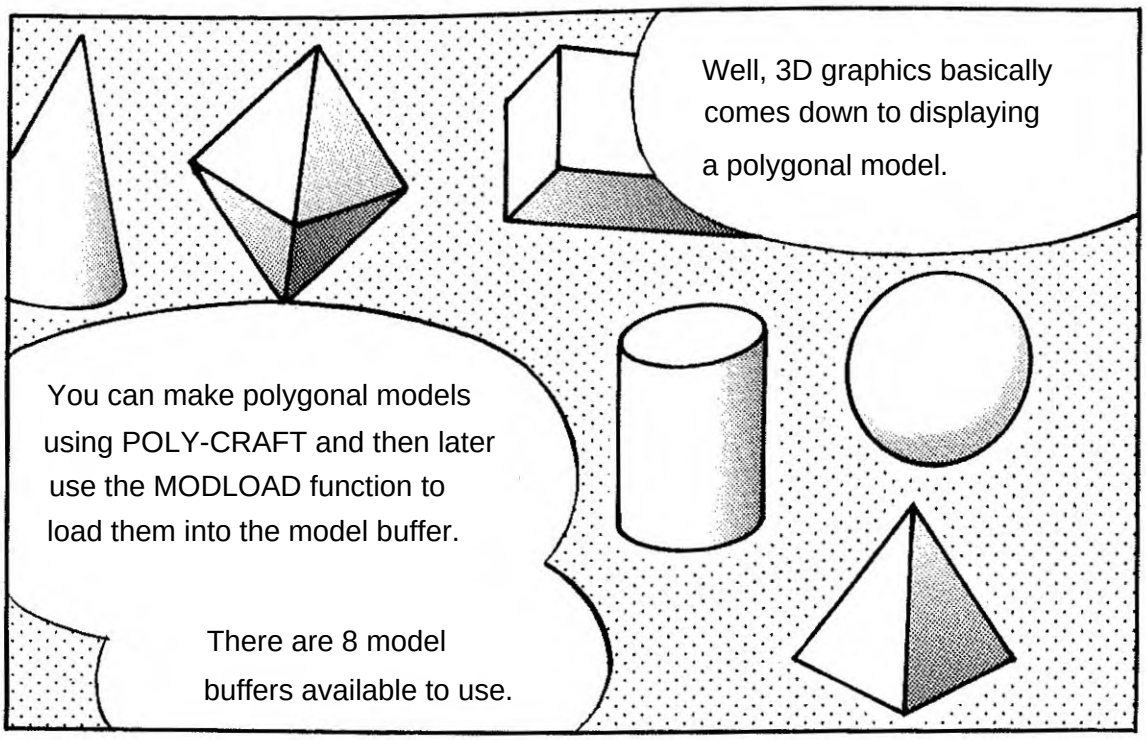

Since there are two coordinates, you can resize the image like so.
Oh, I see.
Start
End
Start
End

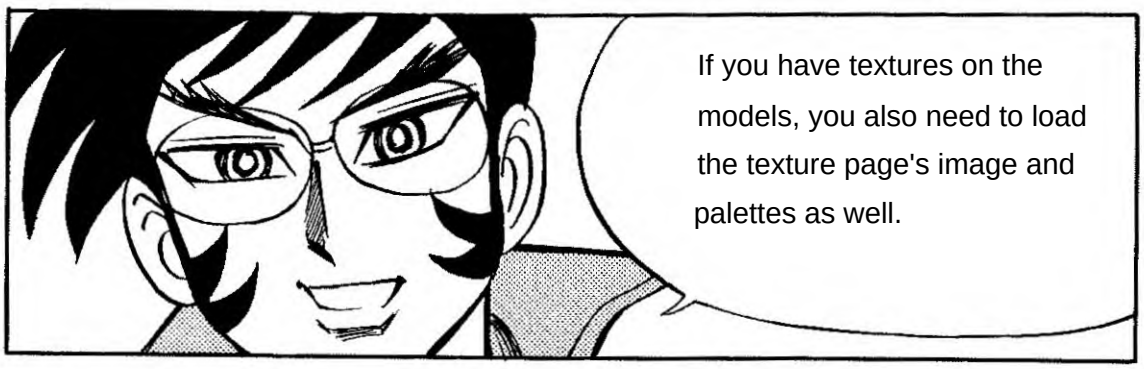
How about rotating?
Sprite primitives cannot be rotated.

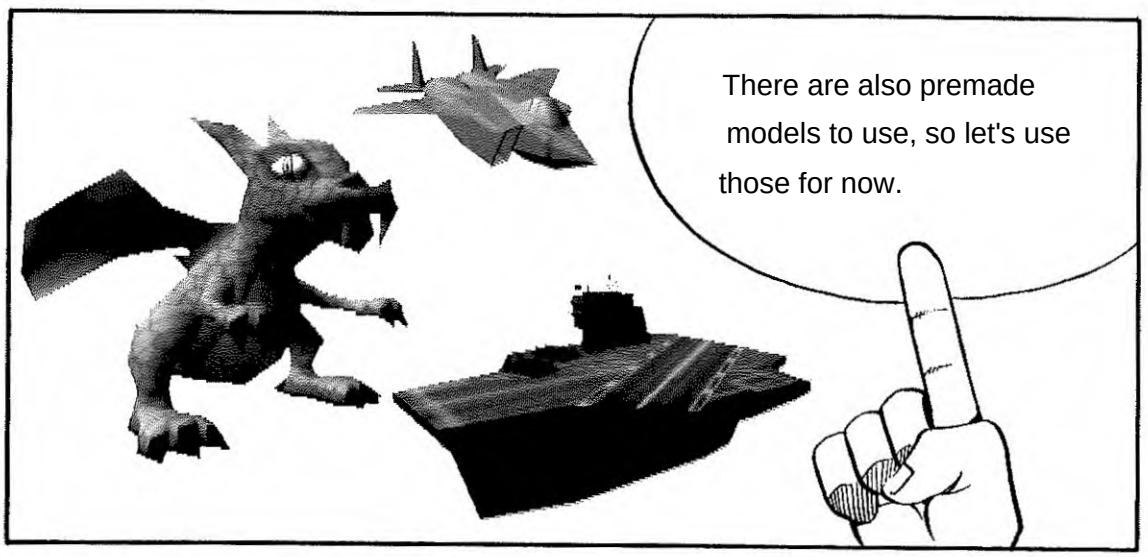

If you want to rotate a sprite, the only way to do it is to place the texture onto a model and rotate the model.
Oh, really?

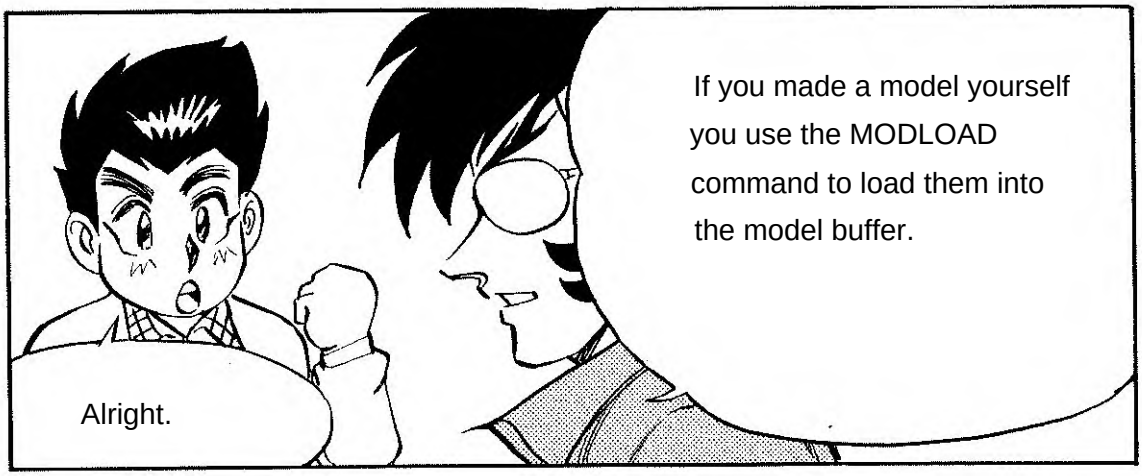

You could also re-draw the sprite on another texture and use a rotated version, I suppose.
Ah, OK!

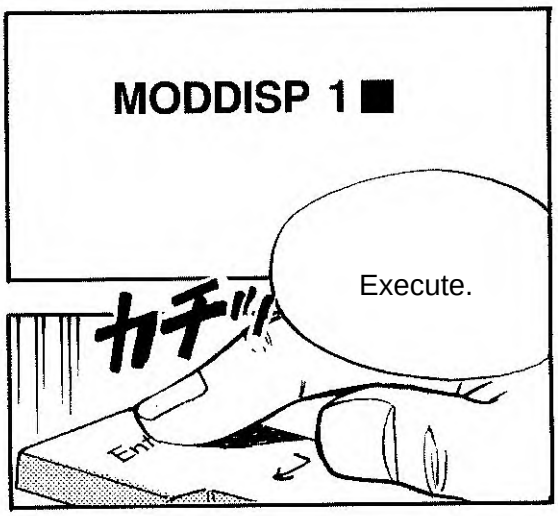

That's about it for 2D graphics.
Next is 3D graphics
Woah, sounds tough!

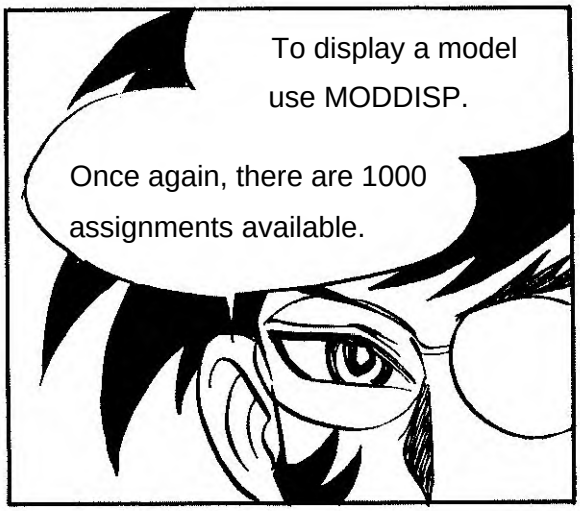

Well, 3D graphics basically comes down to displaying a polygonal model.
You can make polygonal models using POLY-CRAFT and then later use the MODLOAD function to load them into the model buffer.
There are 8 model buffers available to use.

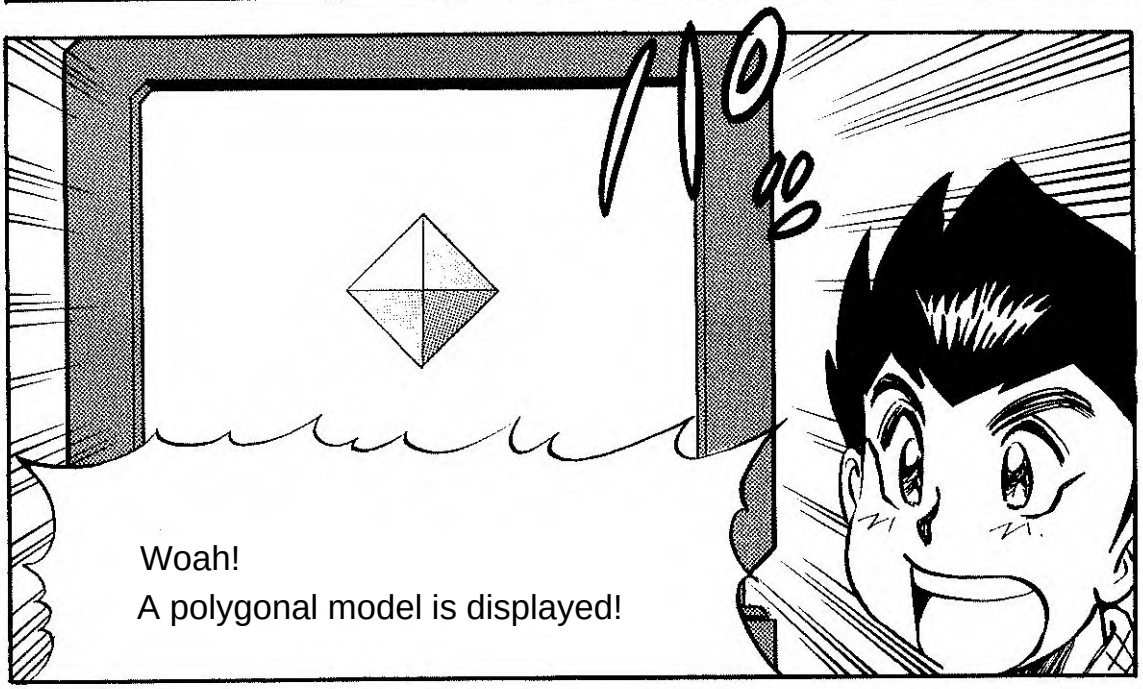

If you have textures on the models, you also need to load the texture page's image and palettes as well.


There are also premade models to use, so let's use those for now.


If you made a model yourself you use the MODLOAD command to load them into the model buffer.
Alright.


MODDISP 1■
Execute.
カチッ!
Ent


To display a model use MODDISP.
Once again, there are 1000 assignments available.


Woah!
A polygonal model is displayed!

If you want to change one model to another, use MODSET.

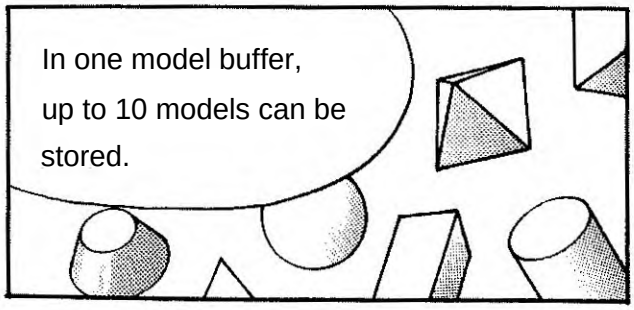
You specify the model number, the buffer number, and the number of the model within the buffer.

Like this.

MODSET Model No., Buffer No., Model number in the buffer

To turn off the display, MODOFF is used, same as other primitives. No explanation needed.

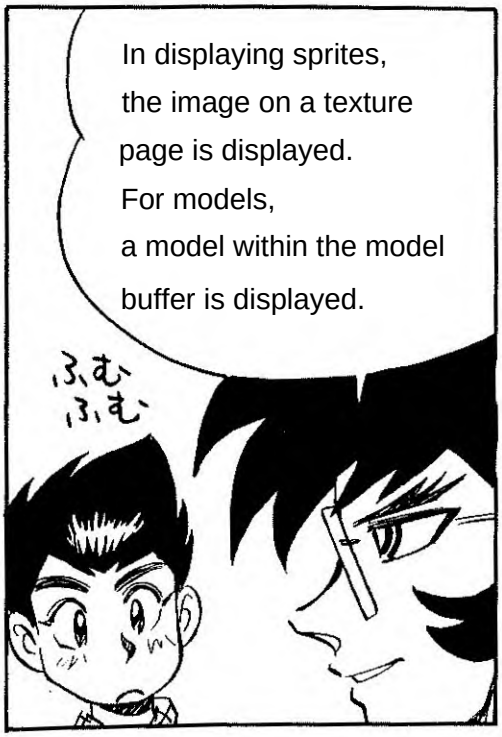
In one model buffer, up to 10 models can be stored.

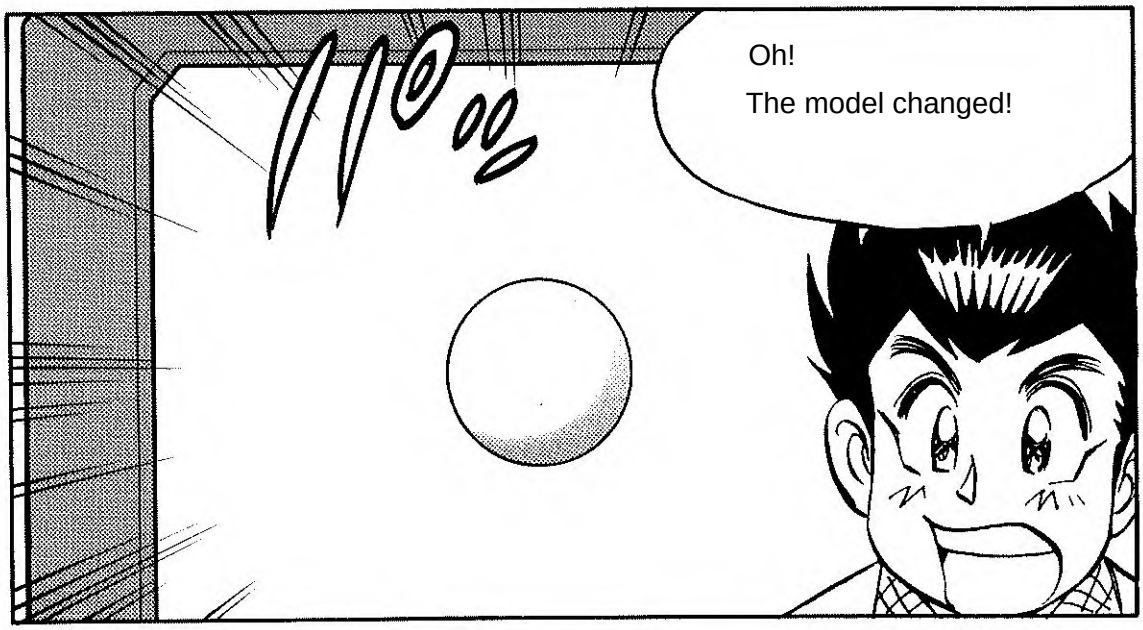


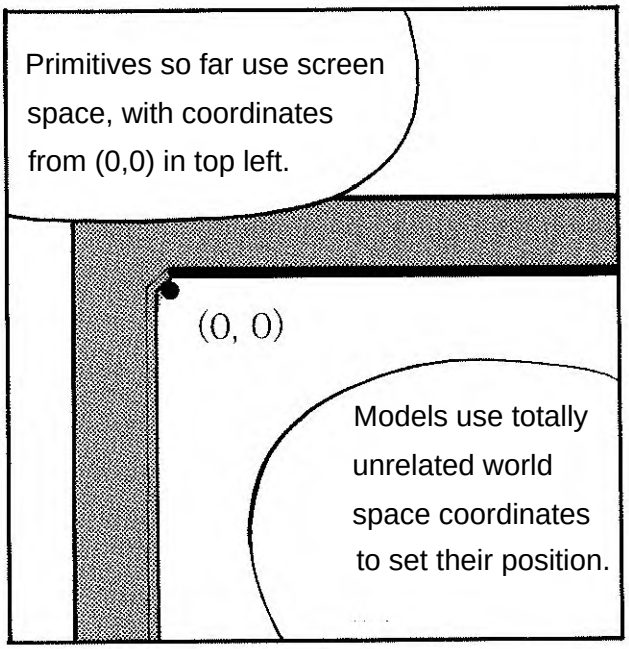
For example, to set model 1 to the tenth model in model buffer 1, it looks like this.

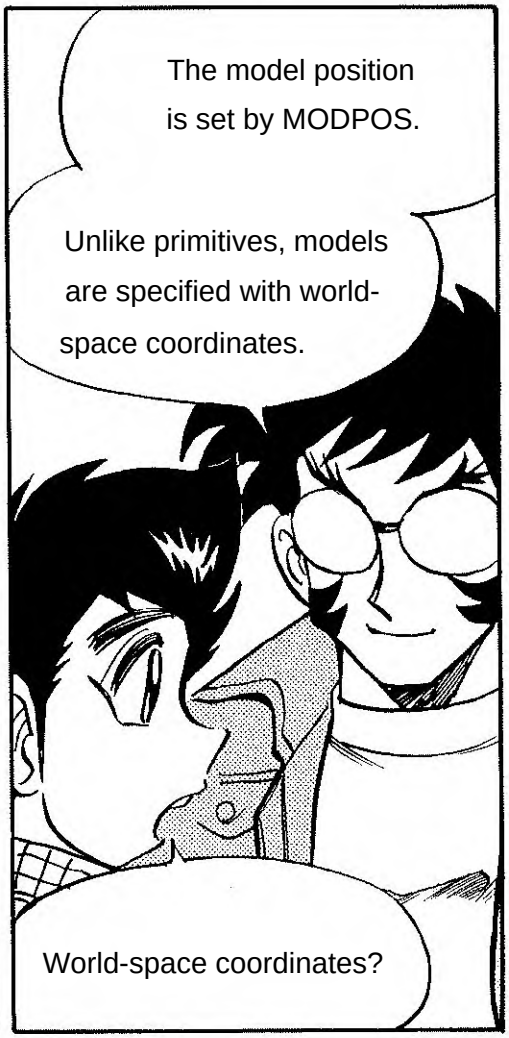
MODESET 1,1,10

In displaying sprites, the image on a texture page is displayed. For models, a model within the model buffer is displayed.

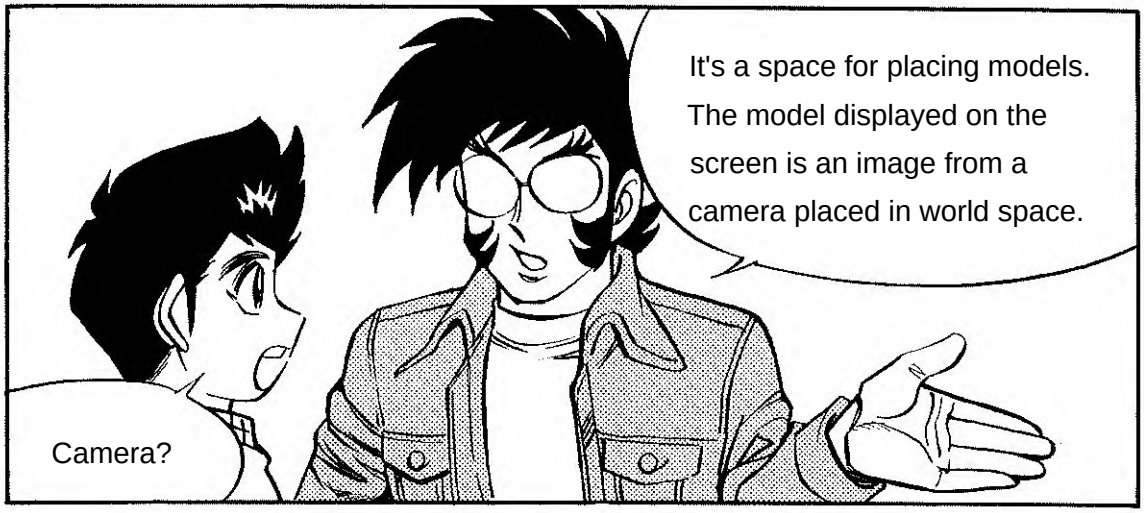


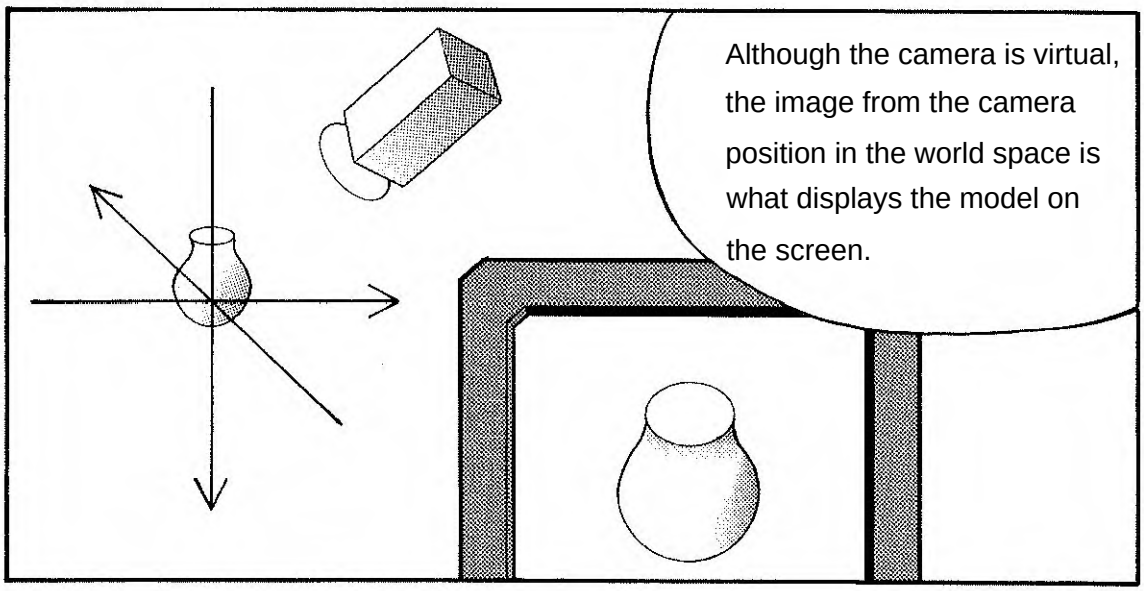

Oh!
The model changed!

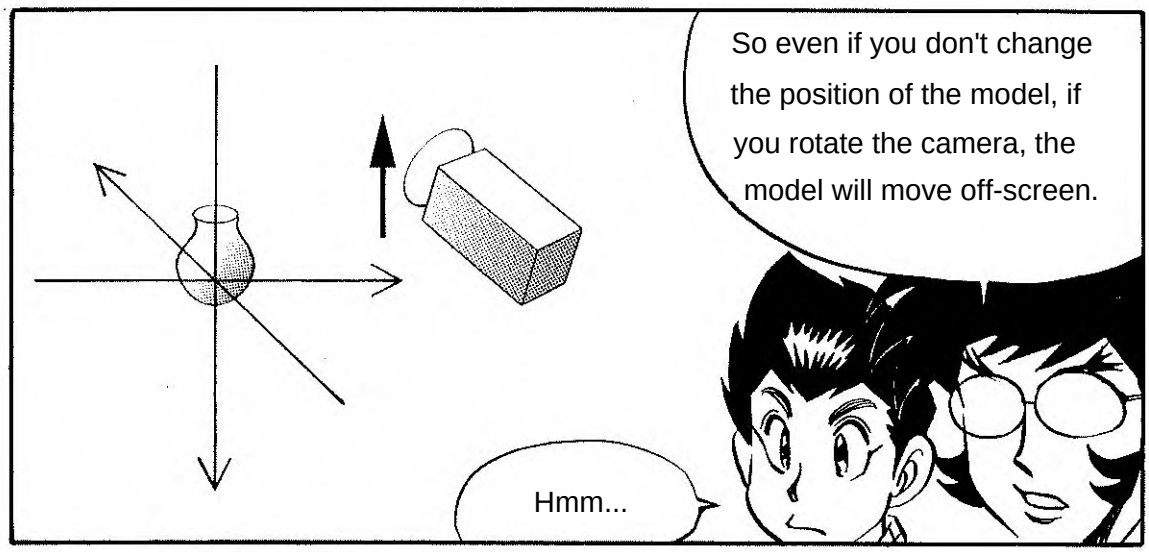

Primitives so far use screen space, with coordinates from (0,0) in top left.
(0, 0)
Models use totally unrelated world space coordinates to set their position.

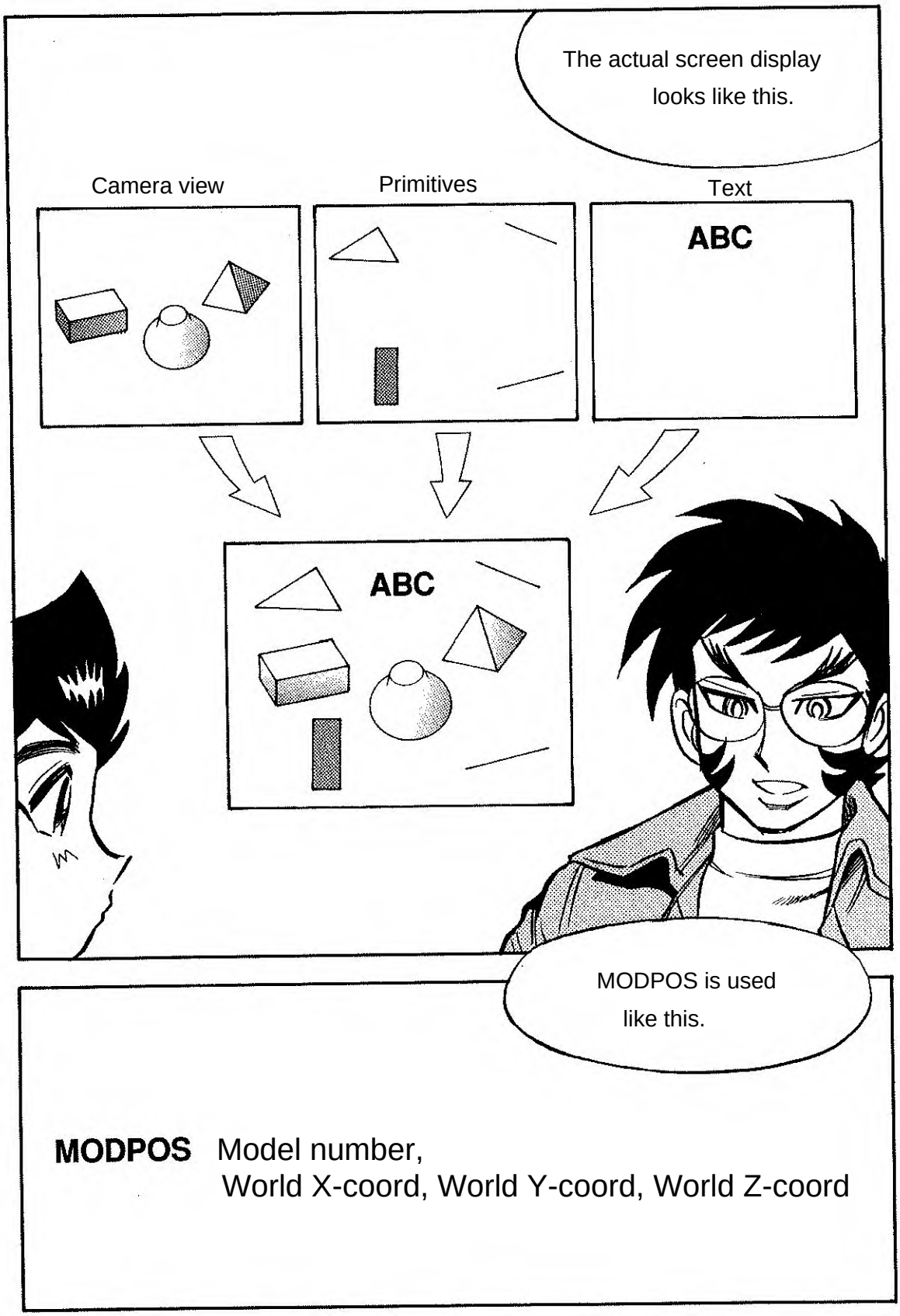

The model position is set by MODPOS.
Unlike primitives, models are specified with world-space coordinates.
World-space coordinates?

Ok so what's world space, then?

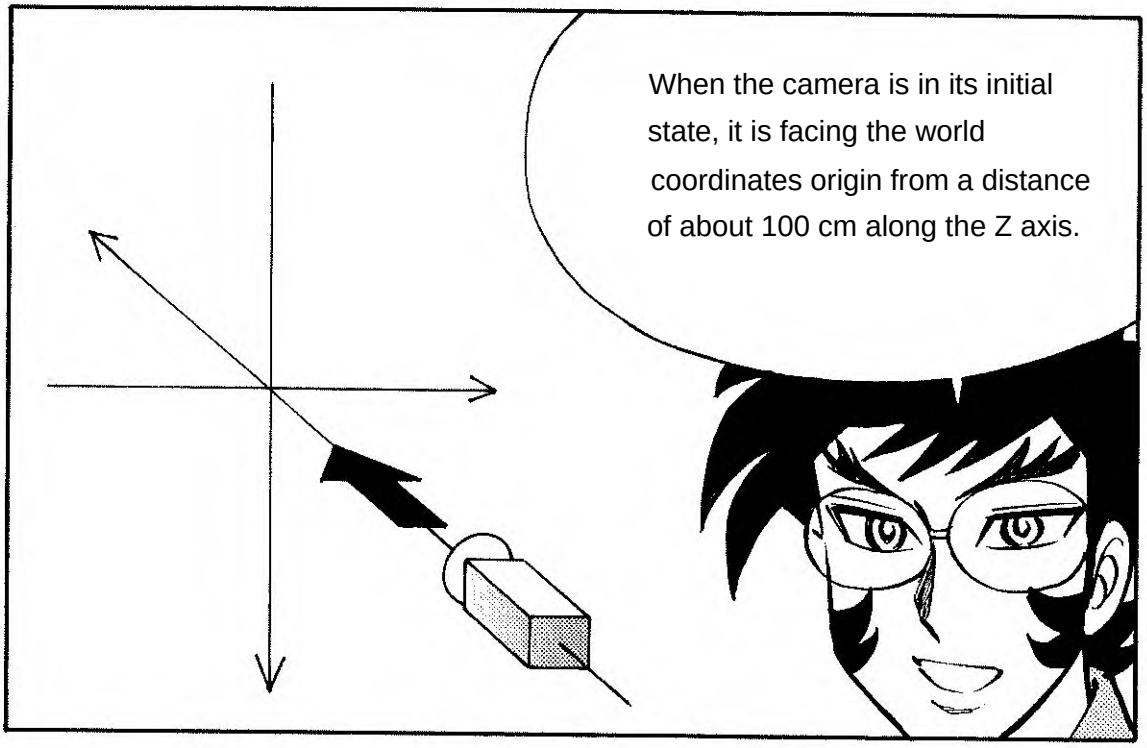

It's a space for placing models. The model displayed on the screen is an image from a camera placed in world space.
Camera?

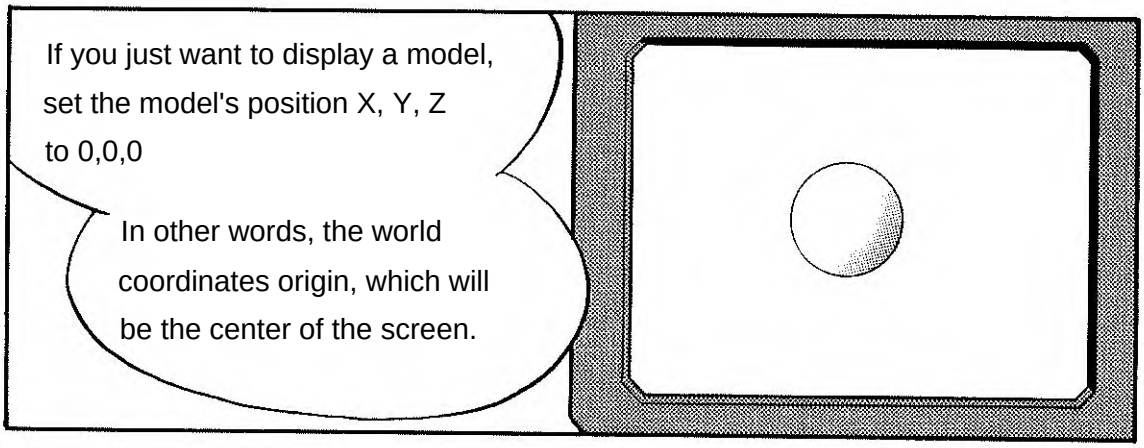

Although the camera is virtual, the image from the camera position in the world space is what displays the model on the screen.

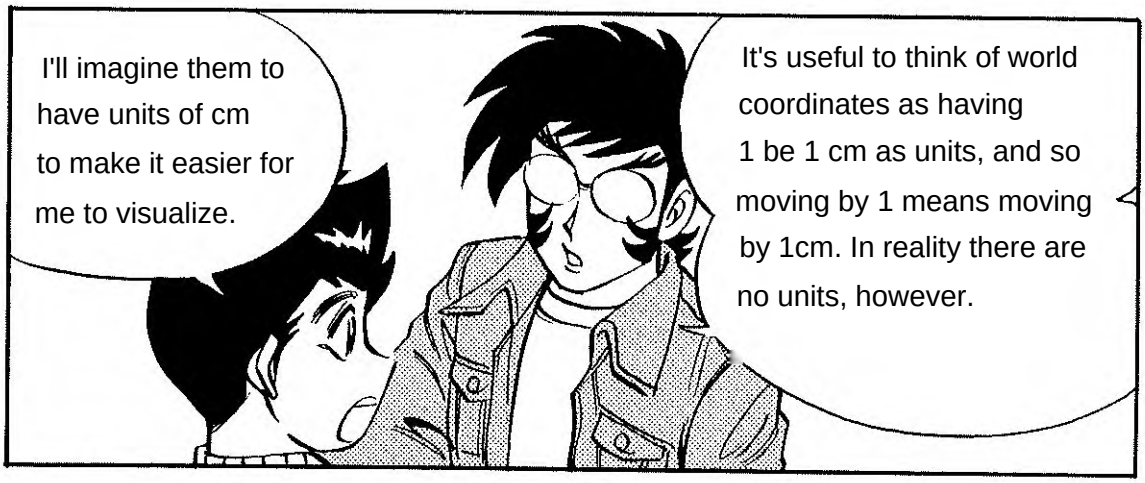

So even if you don't change the position of the model, if you rotate the camera, the model will move off-screen.
Hmm...

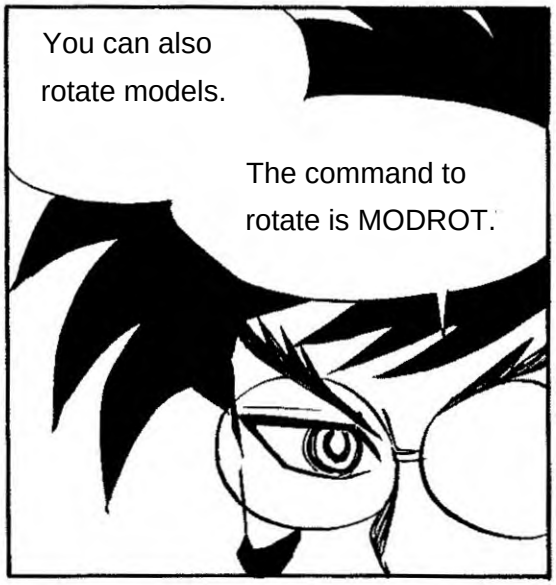


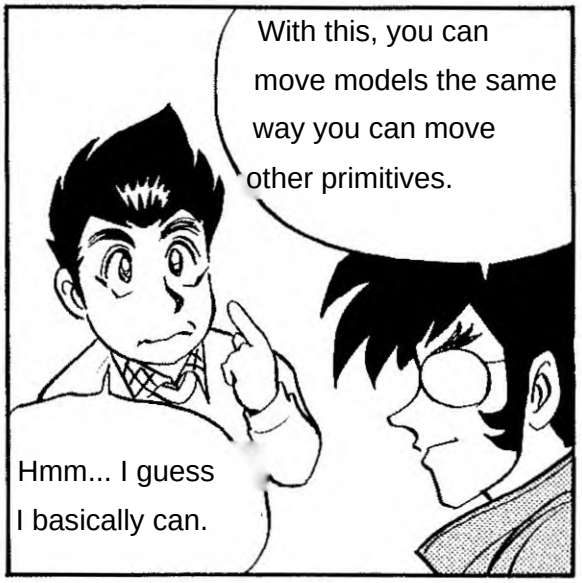
MODPOS Model number,
World X-coord, World Y-coord, World Z-coord


When the camera is in its initial state, it is facing the world coordinates origin from a distance of about 100 cm along the Z axis.

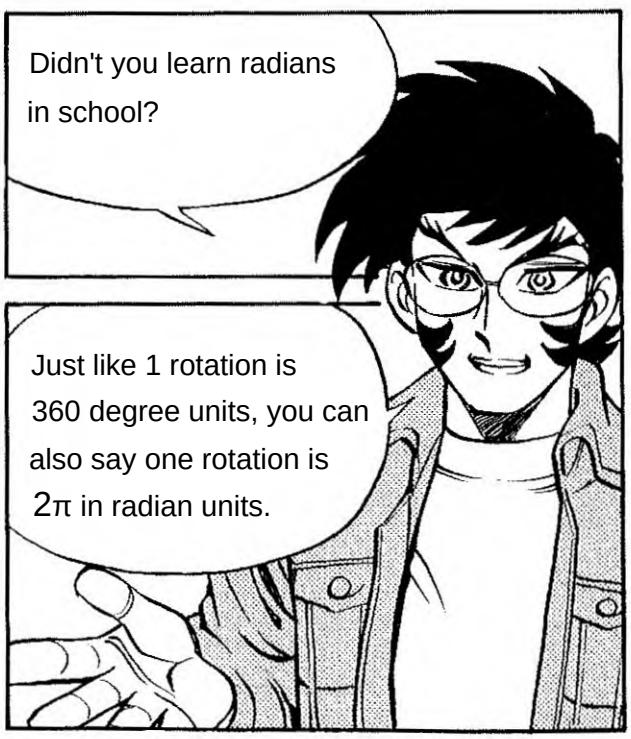

If you just want to display a model, set the model's position X, Y, Z to 0,0,0
In other words, the world coordinates origin, which will be the center of the screen.

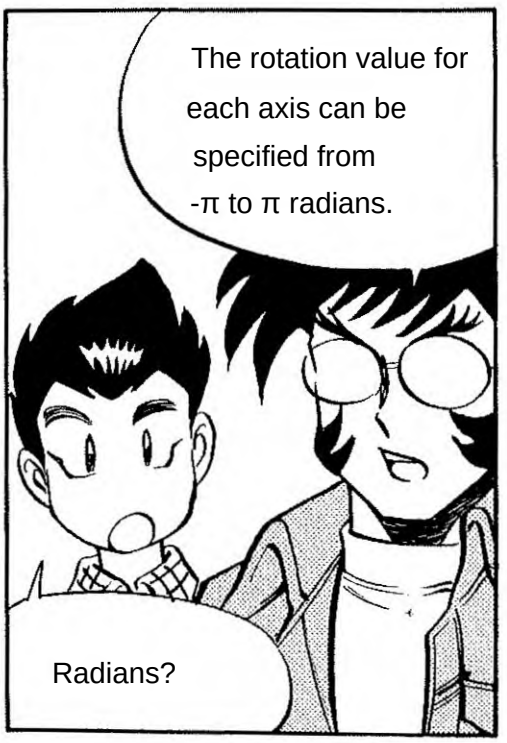

I'll imagine them to have units of cm to make it easier for me to visualize.
It's useful to think of world coordinates as having 1 be 1 cm as units, and so moving by 1 means moving by 1cm. In reality there are no units, however.

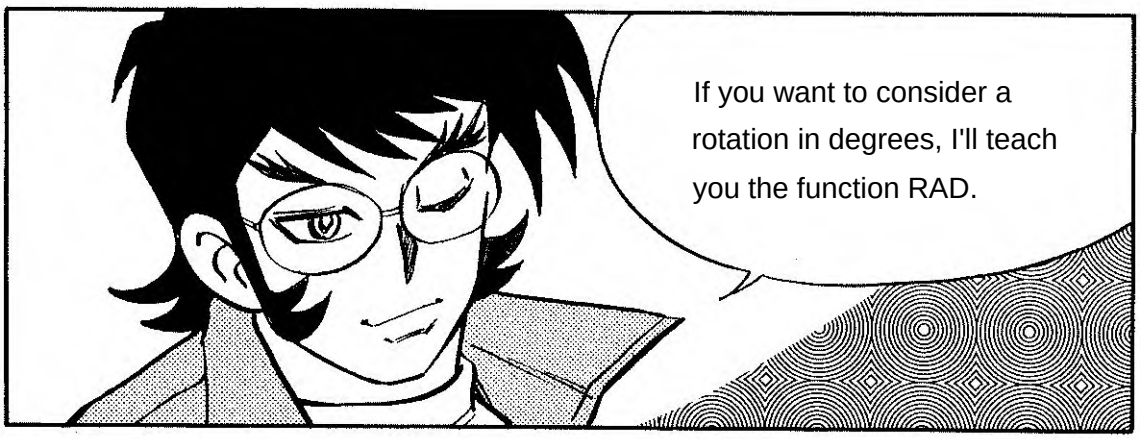


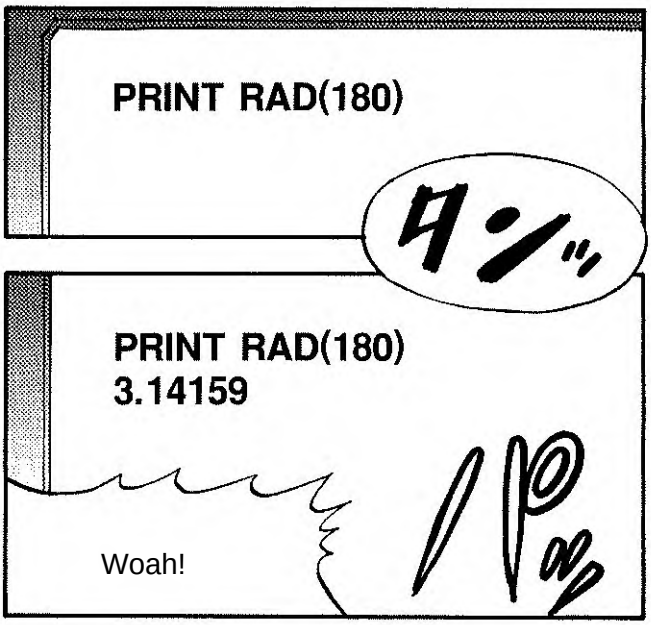


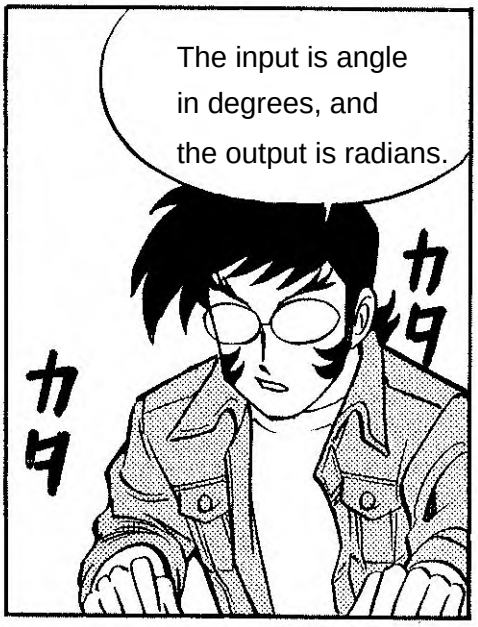
MODROT Model No., X-axis rotation, Y-axis rotation, Z-axis rotation

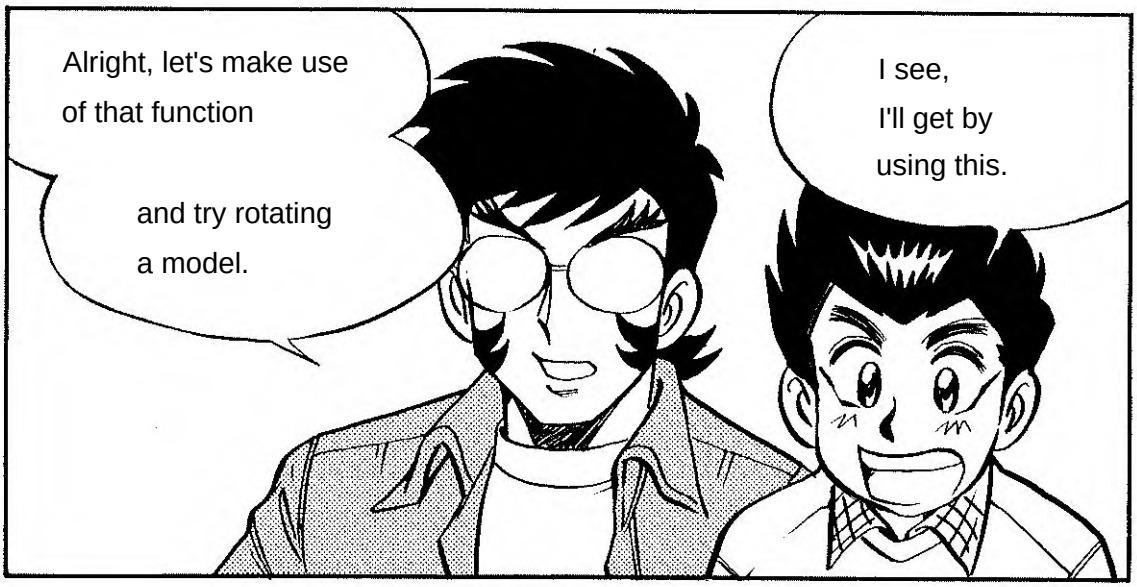

Didn't you learn radians in school?
Just like 1 rotation is 360 degree units, you can also say one rotation is 2π in radian units.




If you want to consider a rotation in degrees, I'll teach you the function RAD.

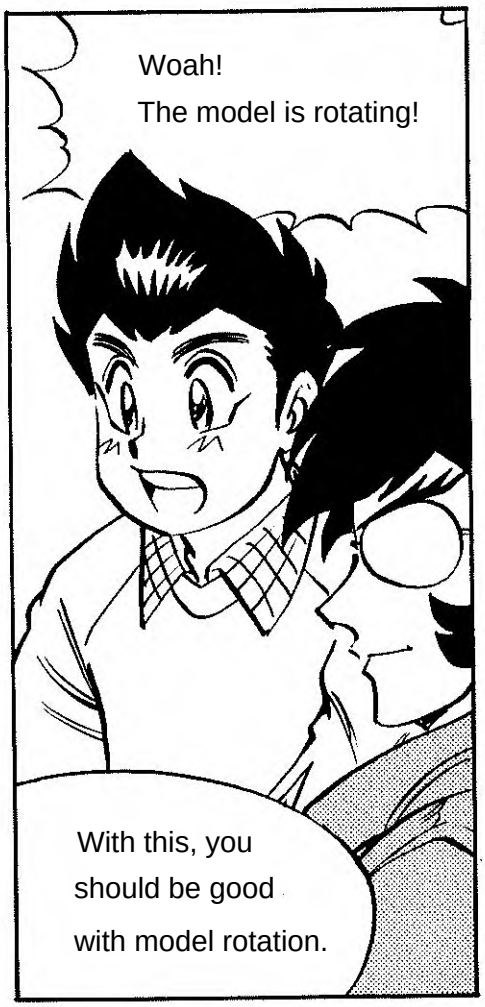

PRINT RAD(180)
PRINT RAD(180)
3.14159
Woah!

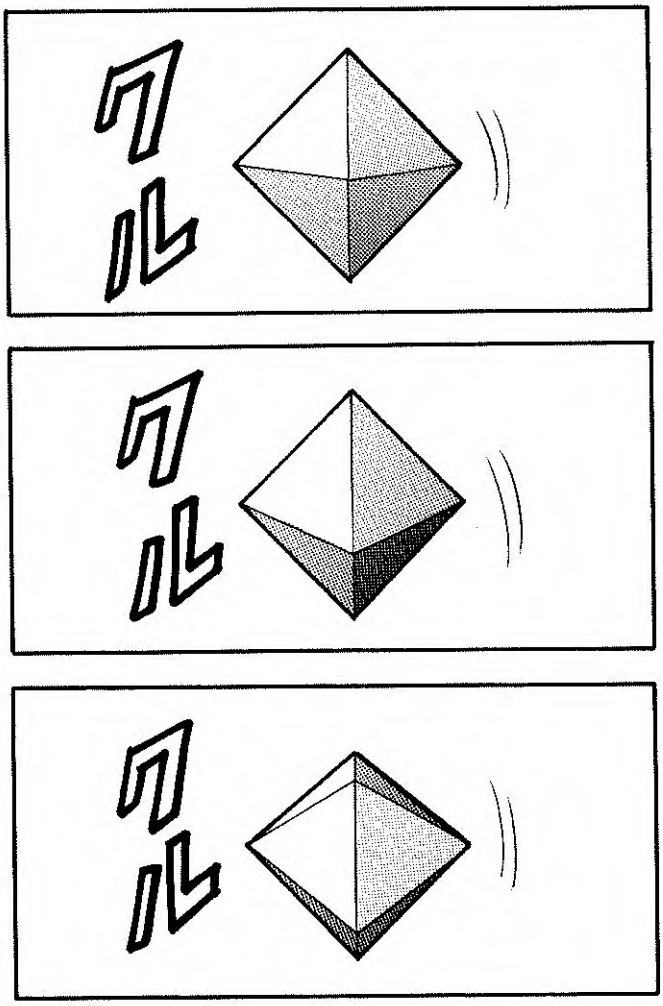

The input is angle in degrees, and the output is radians.

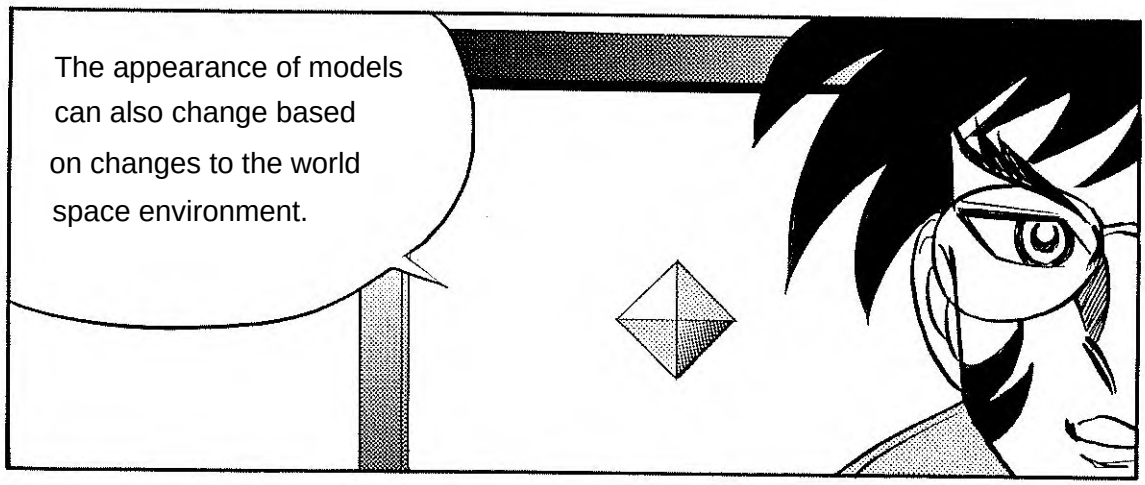

Alright, let's make use of that function
and try rotating a model.
I see, I'll get by using this.

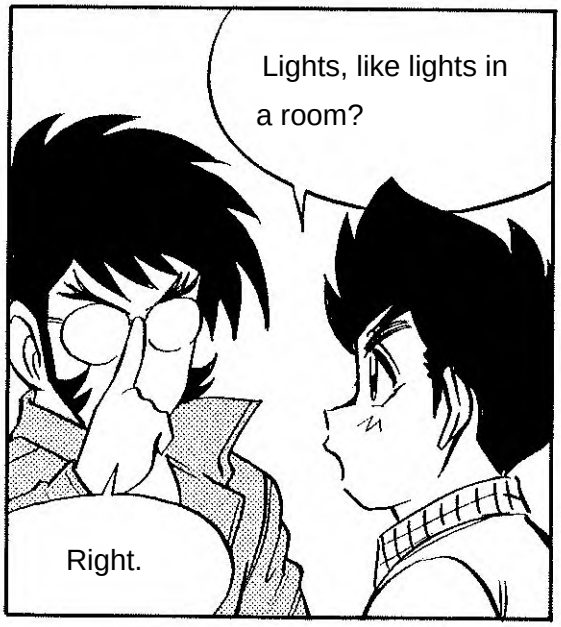
This is a program to rotate a model.

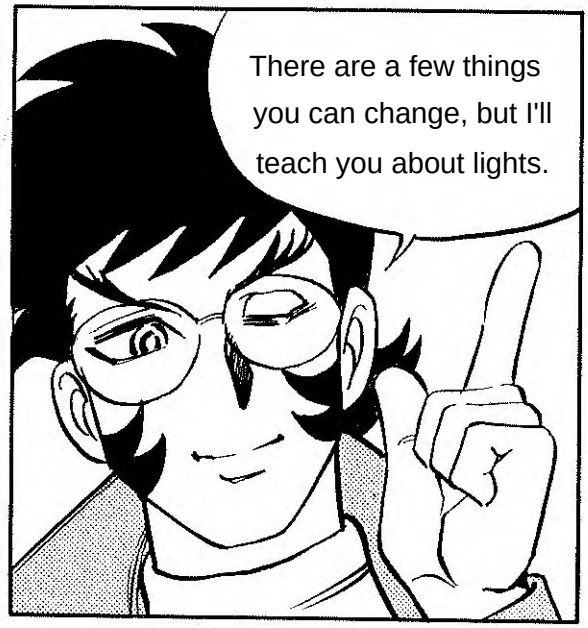
```
10 A=0
20 MODPOS 1,0,0,0
30 MODDISP 1
40 DO
50   MODROT 1,RAD(A),0,0
60   A=A+1
70   IF A>180 THEN A=A-360
80   FSYNC 1
90 LOOP
```

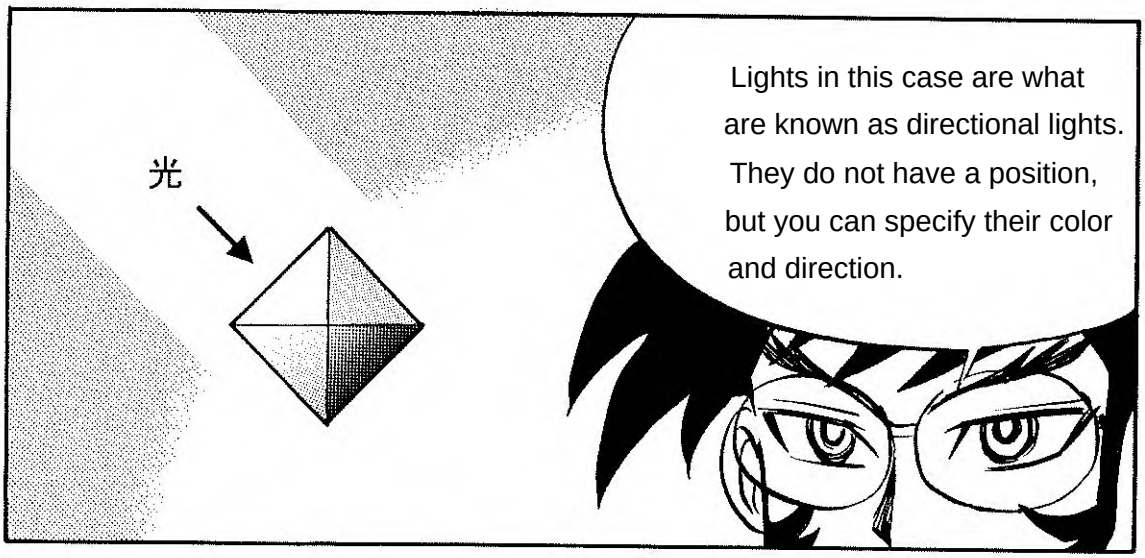


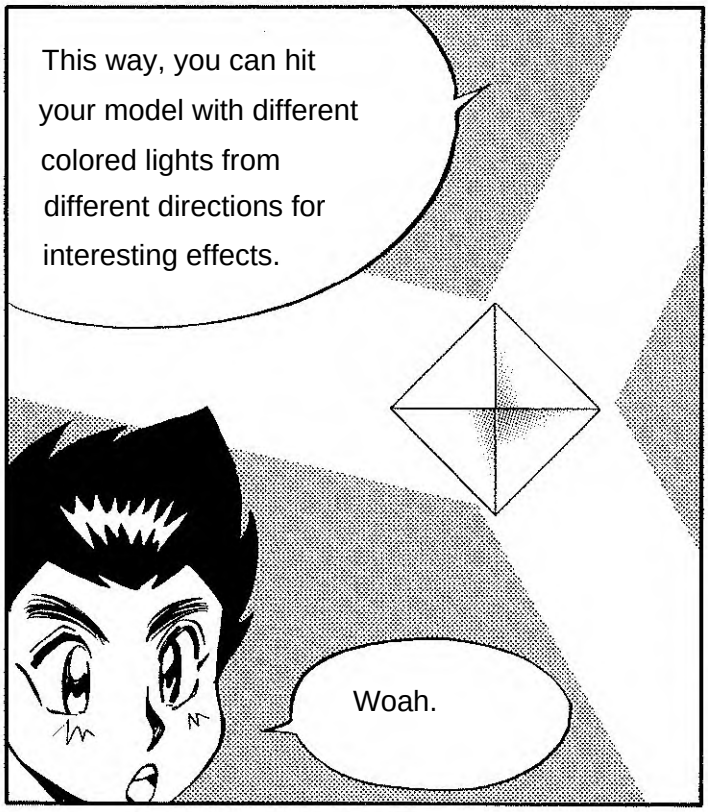


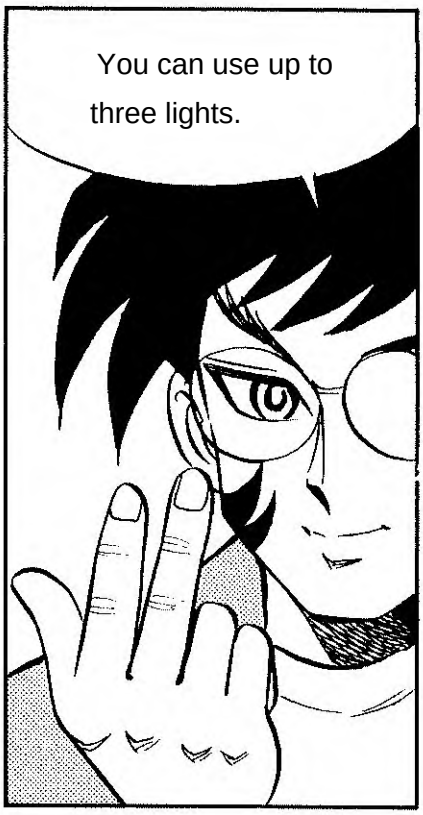

The appearance of models can also change based on changes to the world space environment.

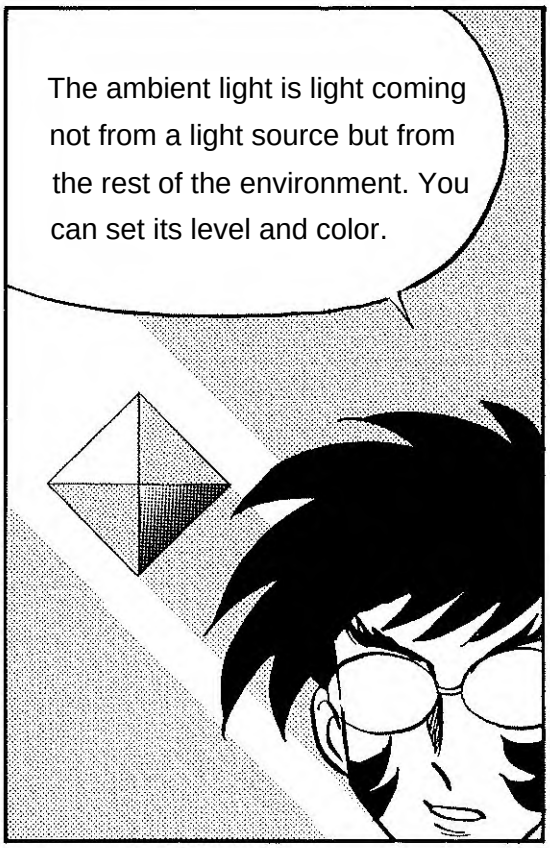

Lights, like lights in a room?
Right.


There are a few things you can change, but I'll teach you about lights.


光
Lights in this case are what are known as directional lights. They do not have a position, but you can specify their color and direction.


This way, you can hit your model with different colored lights from different directions for interesting effects.
Woah.


You can use up to three lights.


The ambient light is light coming not from a light source but from the rest of the environment. You can set its level and color.

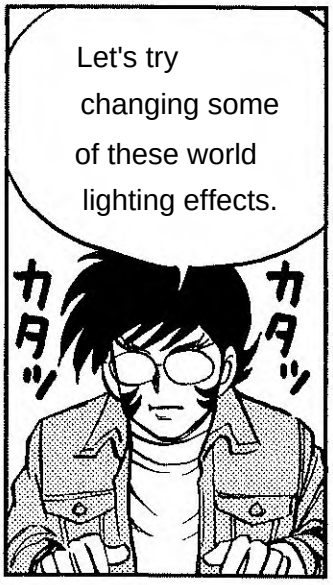
There is also the ambient light.
Ambient?

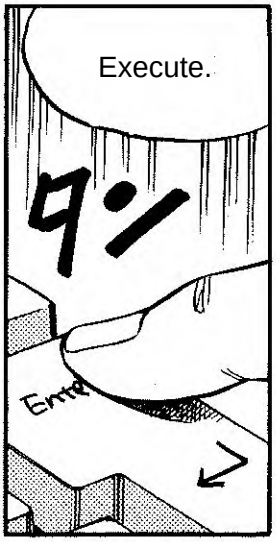
Pitch black...
If you set the ambient light to be zero, parts of the model not directly lit will be pitch black, like it were in scene in a space movie.

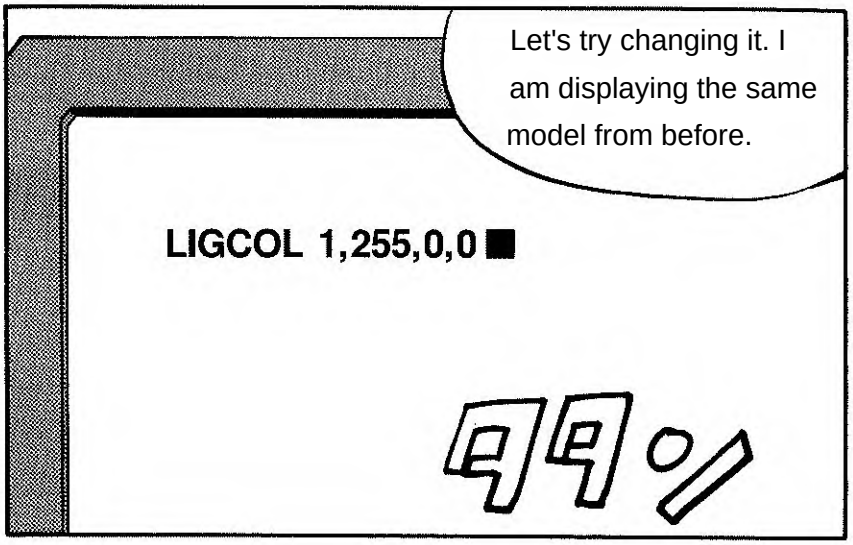
First, with lights, you can change their color with LIGCOL.

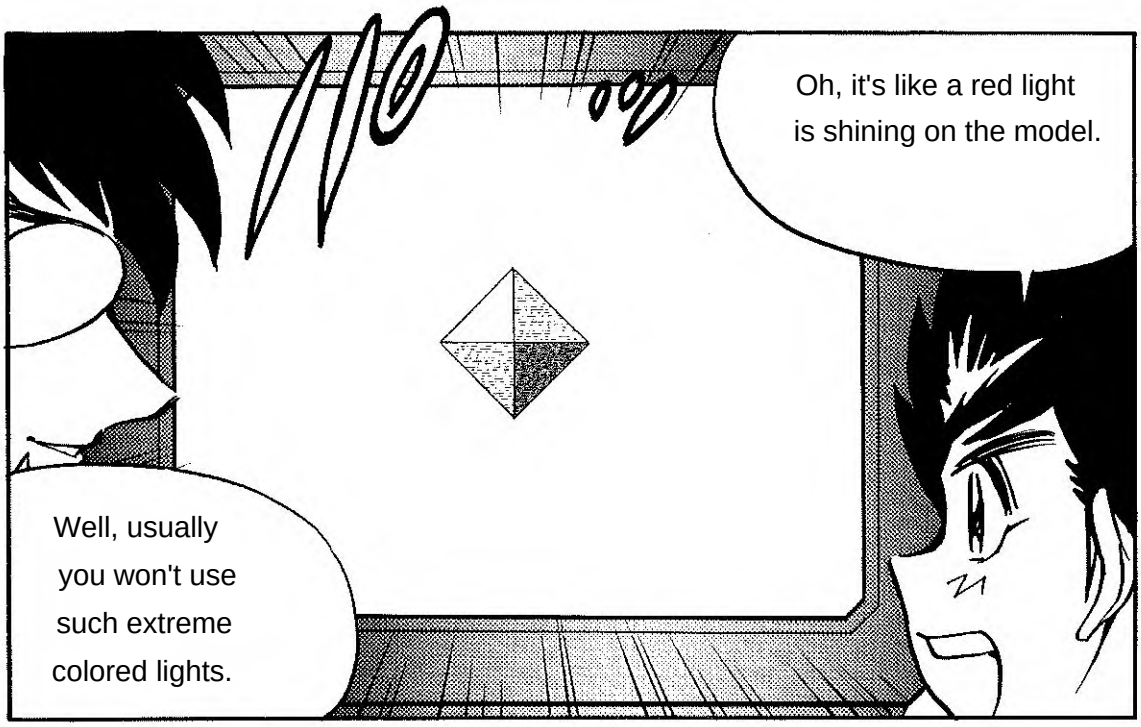
You specify the light number and the RGB value.

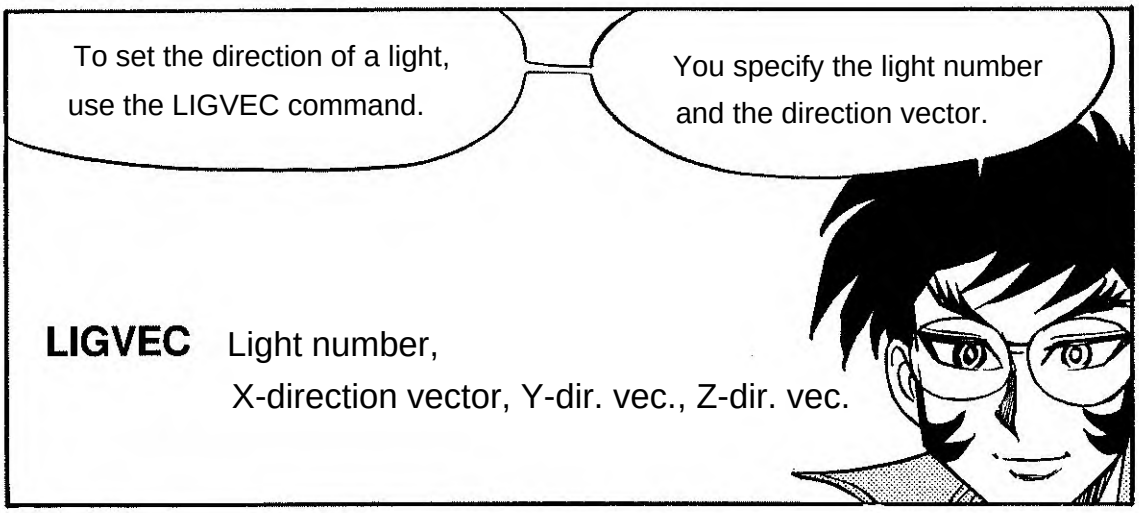
LIGCOL Light number, R, G, B

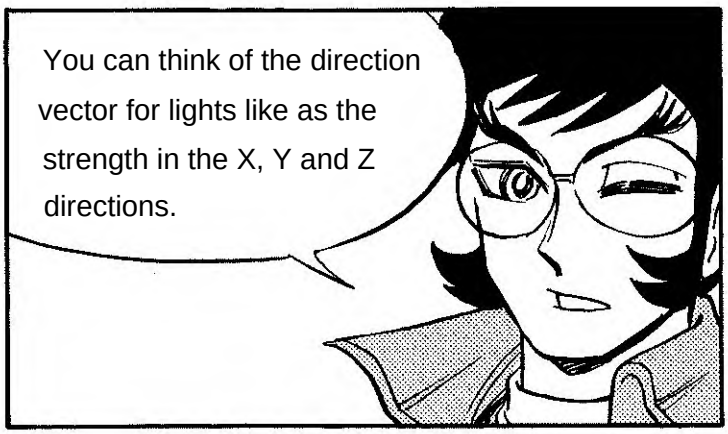




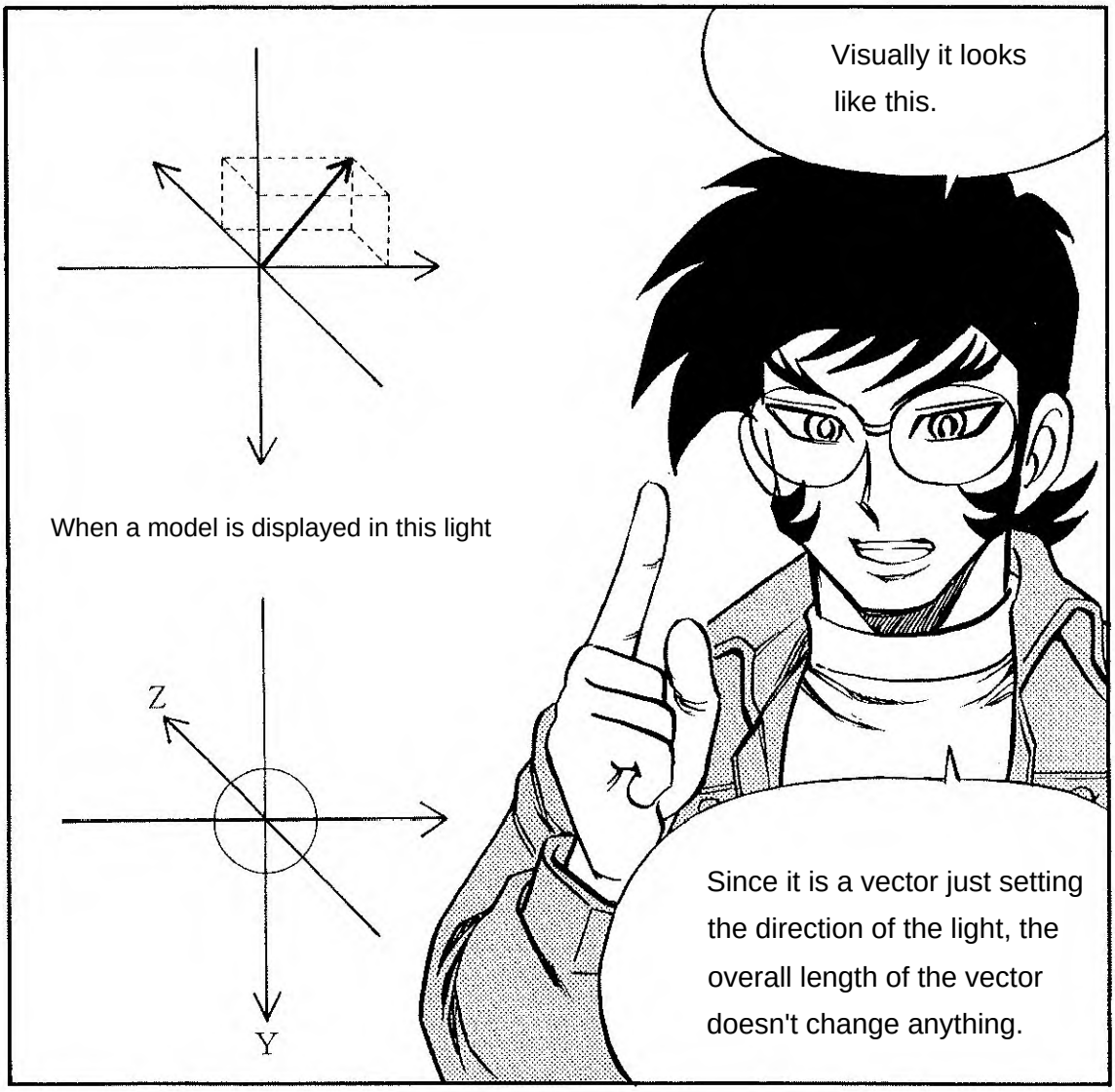


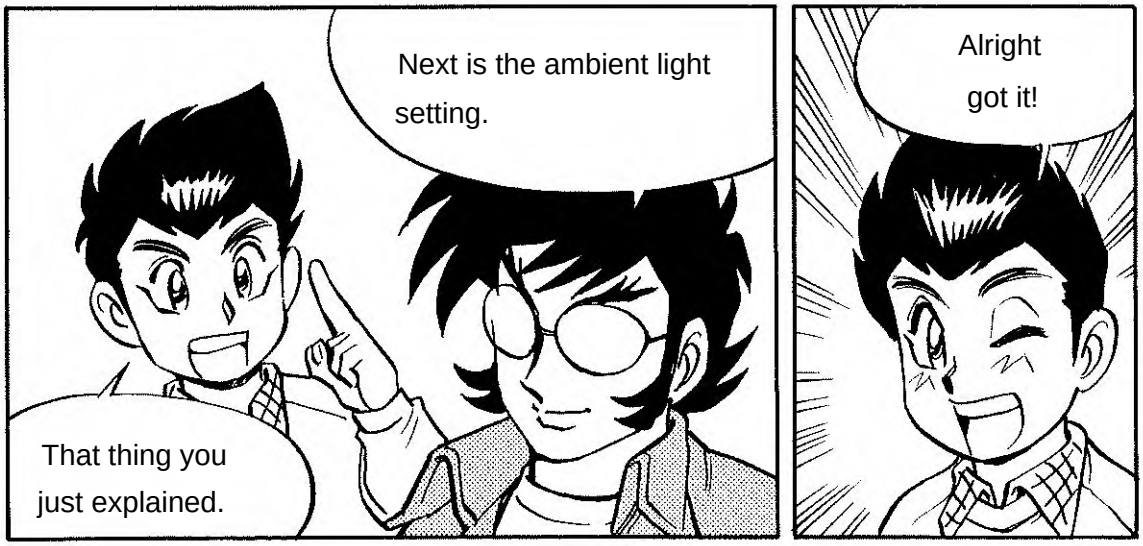

Oh, it's like a red light is shining on the model.
Well, usually you won't use such extreme colored lights.

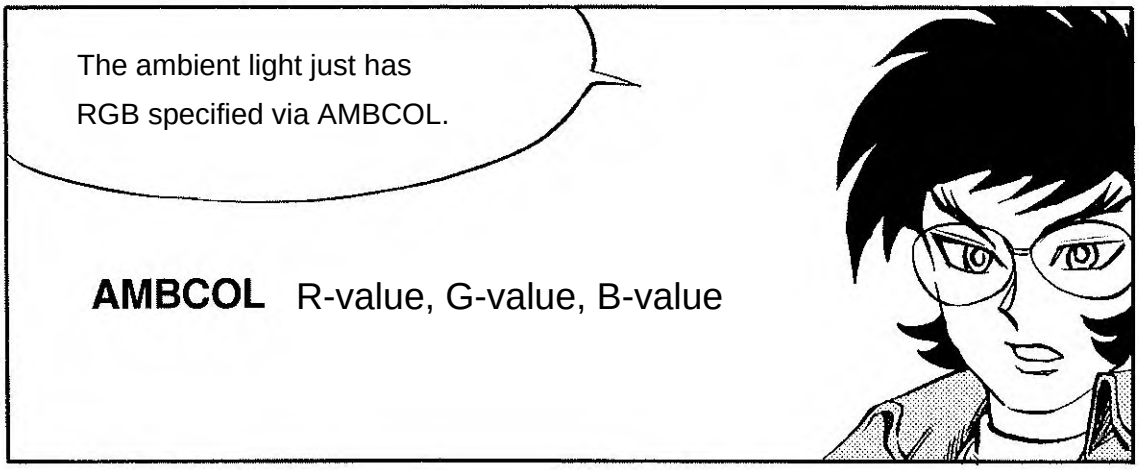

To set the direction of a light, use the LIGVEC command.
You specify the light number and the direction vector.
LIGVEC Light number,
X-direction vector, Y-dir. vec., Z-dir. vec.

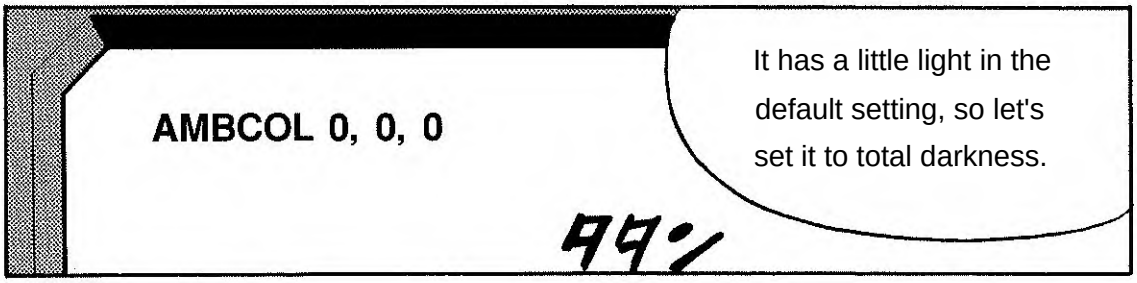

You can think of the direction vector for lights like as the strength in the X, Y and Z directions.

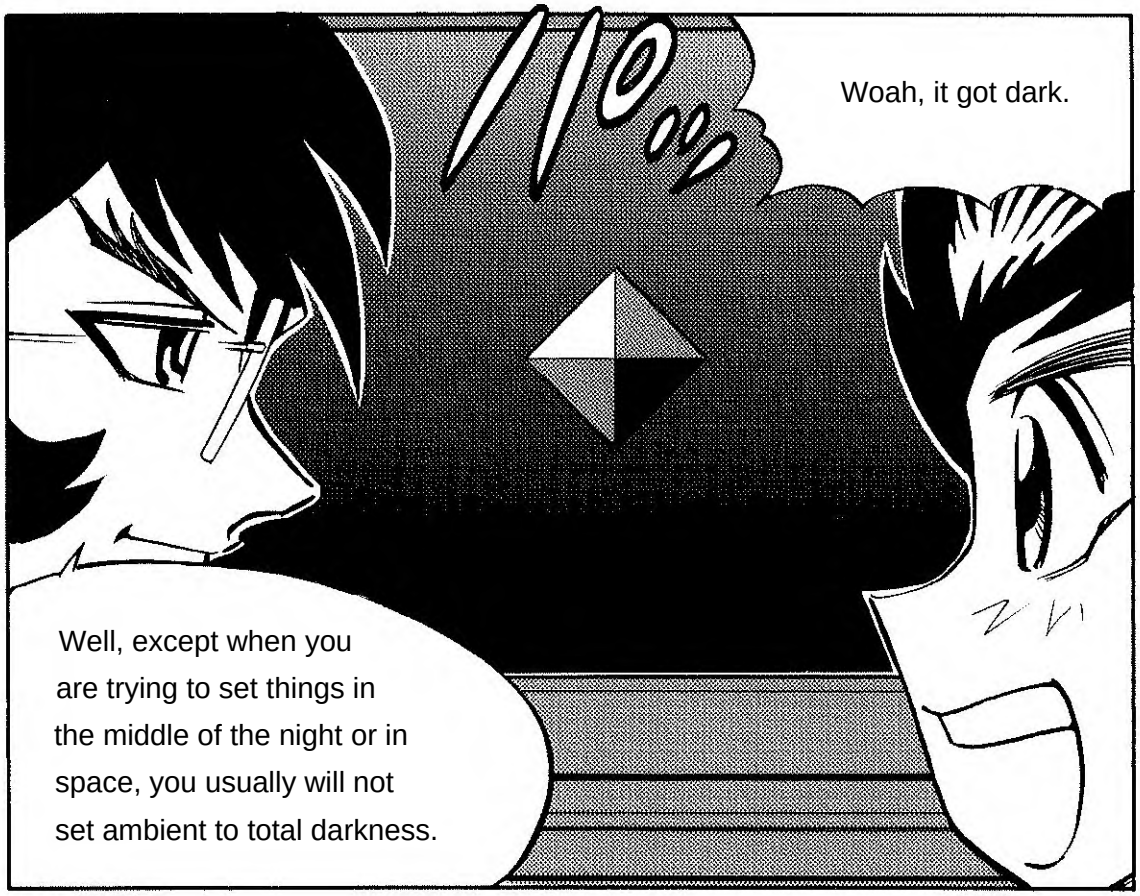
Vector?

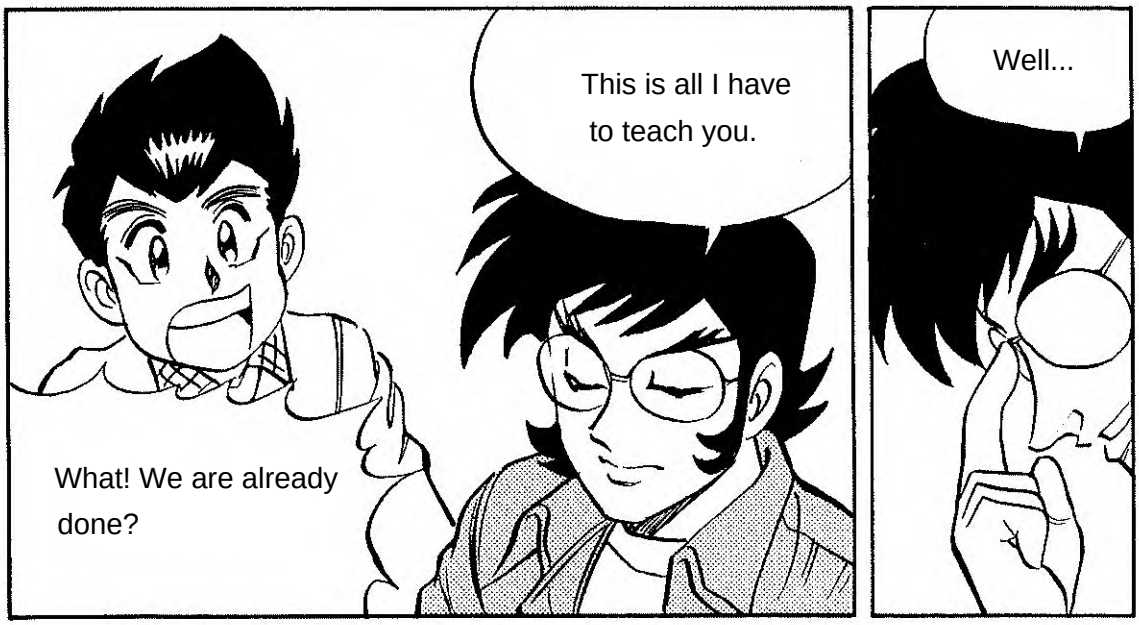

Visually it looks like this.
When a model is displayed in this light
Z
Y
Since it is a vector just setting the direction of the light, the overall length of the vector doesn't change anything.


Next is the ambient light setting.
That thing you just explained.
Alright got it!

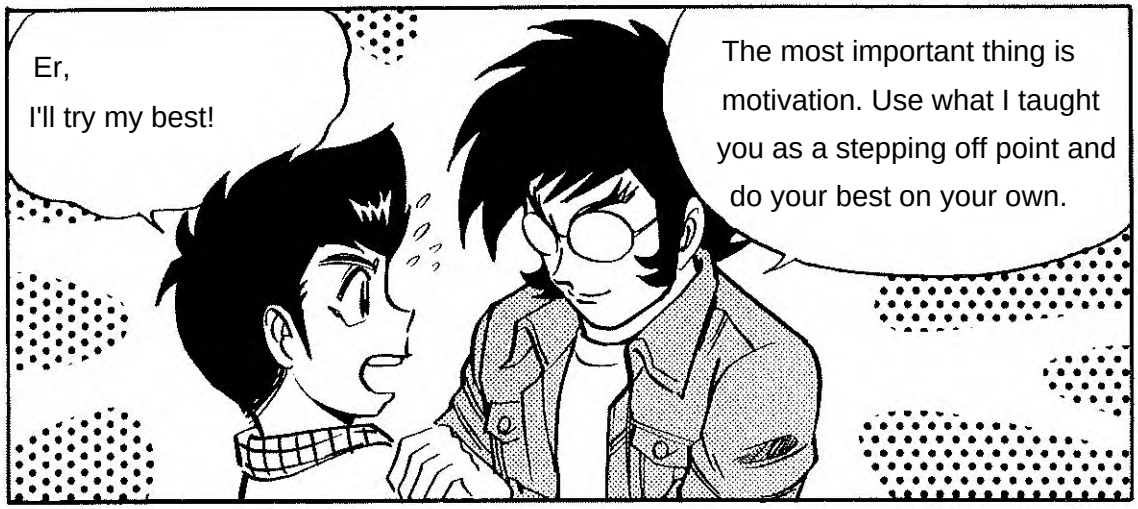

The ambient light just has RGB specified via AMBCOL.
AMBCOL R-value, G-value, B-value


AMBCOL 0, 0, 0
It has a little light in the default setting, so let's set it to total darkness.
99%

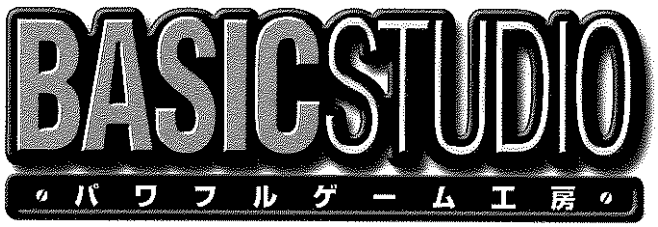

110%
Woah, it got dark.
Well, except when you are trying to set things in the middle of the night or in space, you usually will not set ambient to total darkness.


This is all I have to teach you.
What! We are already done?
Well...

There's still plenty of areas we have yet to touch, but we did all the basics.
For other commands, look them up and learn from the manuals.


Er,
I'll try my best!
The most important thing is motivation. Use what I taught you as a stepping off point and do your best on your own.

You will build skills and confidence by carrying small games one by one to completion.
Don't try to make anything too great from the start. At the beginning just try making something simple.
Alright! Got it!
I'll try my very best!



「はじめよう！EZ-BASIC」

原作　飯塚正樹
作画　すねや　かずみ

一般の単行本のように著作権表記ではないので注意してください。

株式会社アートディンク
〒261-8701千葉市美浜区中瀬2-6WBGマリブウエスト22階
URL http://www.artdink.co jp/

©2001 ARTDINK.All Rights Reserved.
"PlayStation"および“ DUALSHOCK"は株式会社ソニー ・コンピュータエンタテインメントの登録商標です。
ソニー製"プレイステーション2"対応プリンター　　対応。
「popegg」はソニー株式会社の商標です。
その他表記されている商品名、社名は、一般に各社の商標または登録商標です。
2001年4月5日初版発行

SLPS 20084~5